COMPM041: Web Economics - Algorithm Challenge 2016

Sergiu Tripon
Department of Computer Science
University College London
Gower Street, London, WC1E
sergiu.tripon.15@ucl.ac.uk

Santiago Gonzalez
Department of Computer Science
University College London
Gower Street, London, WC1E
hernan.toral.15@ucl.ac.uk

Archie Norman
Department of Computer Science
University College London
Gower Street, London, WC1E
archie.norman.15@ucl.ac.uk

Team Name: teamSSA

1 Introduction

People have experimented with different machine learning algorithms to predict the users click response to each auctioned ad impression in real-time bidding (RTB) display advertising, including Support Vector Machine (SVM), probit regression, decision trees, and gradient boosted decision trees, however logistic regression is used more often. The algorithm is fast, less prone to overfitting and performs as well as SVM.

In order to obtain a good classifier it is important to build a good feature engineering process. The most important feature is historical Click Through Rate (CTR) when such data exists. However, on many websites, new items or documents are introduced daily, which don't have click history. In this case, various other features can be used.

Common features are divided into two categories: item features and user features. Item features include information of a document or a product (e.g. price and posting date etc.). User features include both demographic information and behavioural information such as dwell time and query reformulation. In addition, we can use text features from user queries.

2 Work Assignment

2.1 Sergiu Tripon

Sergiu's main responsabilities were to produce the Statistical Analysis on the training data provided and to work on the Logistic Regression Model. Besides the model itself, Sergiu also performed the model's feature engineering and evaluation and authored the corresponding report sections. The Logistic Regression Model scored an AUC value of 0.80064 on the public scoreboard of the Kaggle competition. He has also briefly experimented with the Vowpal Wabbit library.

2.2 Santiago Gonzalez

Santiago was in charge on performing some experiments with Support Vector Machine and Gradient Boosted Machine models. The created iPython notebook allow the training comparison of a few model setup flavors. He also built and validate some of the feature engineering processes for all implemented models in order to improve the predictions, as well as contributed to the model hyper parameter search process and the evaluation of results. Finally, he also contribute to authoring some of the sections of this report.

2.3 Archie Norman

Archie worked on implementing libFFM, a field aware factorisation model. A main challenge with working on this model has been manipulating the dataset to the necessary format, enabling the model to include each pair of features in individual parameter spaces. Archie has also run models on the Criteo dataset to gain a better understanding of the model.

3 Dataset

The iPinYou training dataset contains 2,847,803 records of processed iPinYou DSP impression logs from some advertising campaigns during three days from the original set of advertising campaigns for offline testing purposes. The following section presents the statistical analysis made on the features contained within it.

4 Statistical analysis

Submission path: *ctr-prediction/analysis/output/*

Table 1 showcases data related to Click. Click 0 indicating the bid request received no clicks has a frequency of 2,845,726, while Click 1, indicating that the bid request received clicks has a frequency of 2,076. This means that 99.92710167350117% of the bid requests received no clicks, while 0.07289832649882261% of the bid requests received clicks.

Table 1: Click CTR and frequency, sorted by CTR.

Click	**Frequency**
0	2845726
1	2076

Table 2 showcases Top 5 User Agent, sorted by CTR. ios_other has the highest CTR and the lowest frequency. android_safari is the most frequent but third in terms of CTR.

Table 2: Top 5 User Agent CTR and frequency, sorted by CTR.

User Agent	**CTR**	**Frequency**
ios_other	0.0666666666667	15
android_chrome	0.0108303249097	277
android_safari	0.0106951871658	13090
android_other	0.00734522560336	953
mac_other	0.00653976410137	8563

Table 3 showcases Top 5 Hour, sorted by CTR. Hour 23 has the highest CTR and the second lowest frequency. Hour 0 is the most frequent but third in terms of CTR.

Table 3: Top 5 Hour CTR and frequency, sorted by CTR.

Hour	CTR	Frequency
23	0.000986636069863	180411
18	0.0009643824721	201165
0	0.00089908749329	223560
14	0.000802004458039	180797
20	0.000783891039146	163288

Table 4 showcases Top 5 Weekday, sorted by CTR. Weekday 5 has the highest CTR. Hour 1 is the least frequent and last in terms of CTR. Weekday 4 is the most frequent and has the second highest CTR.

Table 4: Top 5 Weekday CTR and frequency, sorted by CTR.

Weekday	CTR	Frequency
5	0.000903535496219	392901
4	0.000834394518222	412275
3	0.000772489021842	385766
2	0.000682842445486	395406
1	0.000679142968773	381363

Table 5 showcases Top 5 City, sorted by CTR. City 359 is the least frequent and has the highest CTR. City 300 is the most frequent but third in terms of CTR.

Table 5: Top 5 City CTR and frequency, sorted by CTR.

City	CTR	Frequency
359	0.0059880239521	167
323	0.00520833333333	192
300	0.00432098765432	1620
398	0.00409836065574	244
209	0.003003003003	1332

Table 6 showcases Top 5 Region, sorted by CTR. Region 394 has the highest CTR. City 395 is the least frequent and second in terms of CTR. City 298 is the most frequent but has the third highest CTR.

Table 6: Top 5 Region CTR and frequency, sorted by CTR.

Region	CTR	Frequency
394	0.00182648401826	5475
395	0.00139470013947	717
298	0.00123148610418	30045
253	0.0011219986536	13369
393	0.00110698189294	12647

Table 7 showcases data related to Ad exchange, sorted by CTR (Click Through Rate). There are 3 Ad Exchanges, with Ad Exchange 2 being the most frequent in the data set, while Ad Exchange 1 has the highest CTR.

Table 7: Ad exchange sorted by CTR.

Ad exchange	CTR	Frequency
1	0.00127279935376	670962
2	0.000574171656299	1236564
3	0.000544520970438	940276

Table 8 showcases the Top 5 Ad slot floor prices, sorted by CTR. The Ad slot floor price of 85 has the highest CTR but also the least frequency. The Ad slot floor price of 280 is the most frequent, but third in terms of CTR.

Table 8: Top 5 Ad slot floor price sorted by CTR.

Ad slot floor price	CTR	Frequency
85	0.125	8
154	0.0240963855422	83
280	0.0.00191662673694	2087
192	0.00189933523267	1053
125	0.00151171579743	1323

Table 9 showcases data related to Ad slot visibility, sorted by CTR. Ad slot visibility 1 has the highest CTR, while Ad slot visibility 255 is second in terms of CTR. Ad slot visibility 0 is third and Ad slot visibility 2 is fourth.

Table 9: Ad slot visibility sorted by CTR.

Ad slot visibility	CTR
1	0.00163883898031
255	0.0012573584043
0	0.000645349346314
2	0.00042412373941

Table 10 showcases data related to Ad slot format, sorted by CTR. Ad slot format 0 is the most frequent but also has the lowest CTR, while Ad slot format 5 is the least frequent but has the highest CTR.

Table 10: Ad slot format sorted by CTR.

Ad slot format	CTR	Frequency
5	0.00997112737528	529786
1	0.00086871623392	641176
0	0.000561364179269	2176840

Table 11 showcases the Top 5 Ad slot width, sorted by CTR. The Ad slot width of 300 pixels is the most frequent and has the highest CTR. The Ad slot width of 120 pixels is the least frequent and second in terms of CTR.

Table 12 showcases data related to Ad slot height, sorted by CTR. The Ad slot height of 250 pixels has the highest CTR. The Ad slot height of 200 pixels is the most frequent but last in terms of CTR. The Ad slot height of 60 pixels is the least frequent and fourth in terms of CTR.

Table 11: Top 5 Ad slot width sorted by CTR.

Ad slot width	CTR	Frequency
300	0.00109643572226	928463
120	0.00104336485165	15335
1000	0.000680588525303	789023
160	0.000634253302648	110366
336	0.000609699524169	150894

Table 12: Ad slot height sorted by CTR.

Ad slot height	CTR	Frequency
250	0.0010105417882	1044984
600	0.000684163212703	125701
280	0.000609699524169	150894
60	0.000578398867991	96819
90	0.000570717645816	1343922
200	0.000222269015699	85482

Table 13 showcases Top 5 Ad slot width-height, sorted by CTR. The Ad slot width-height of 300 and 250 pixels is the most frequent and has the highest CTR. The Ad slot width-height of 120 and 600 pixels is the least frequent but second in terms of CTR.

Table 13: Top 5 Ad slot width-height sorted by CTR.

Ad slot width	Ad slot height	CTR	Frequency
300	250	0.00109643572226	928463
120	600	0.00104336485165	15335
1000	90	0.000680588525303	789023
160	600	0.000634253302648	110366
336	280	0.000609699524169	150894

Table 14 showcases data related to Advertiser ID. There is only one Advertiser ID, indicating that there is only one advertiser. The ID is 3386, has a CTR of 0.000728983264988 and its frequency is 2,847,802.

Table 14: Advertiser ID CTR and frequency, sorted by CTR.

Advertiser ID	CTR	Frequency
3386	0.000728983264988	2847802

Table 15 showcases data related to Anonymous URL. There is only one Anonymous URL record, with value ”null” and has a CTR of 0.000728983264988, while its frequency is 2,847,802.

Table 15: Anonymous URL ID CTR and frequency, sorted by CTR.

Anonymous URL ID	CTR	Frequency
null	0.000728983264988	2847802

Table 16 showcases data related to Log Type. There is only one Log Type record, with value ”1” and has a CTR of 0.000728983264988, while its frequency is 2,847,802.

Table 16: Log Type CTR and frequency, sorted by CTR.

Log Type	**CTR**	**Frequency**
1	0.000728983264988	2847802

5 Dataset partition

In order to train our candidate models, we split the data set randomly in 90% training and 10% validation, with a seed value of 1337 to allow our implementation to train and evaluate the different models with the same partition.

6 Feature Engineering

6.1 Feature Selection

The initial feature engineering process consisted in selecting relevant features. A number of columns:

1. *user_id*
2. *advertiser_id*
3. *anonymous_url_id*

were deemed irrelevant. ***user_id*** is unique to a specific user which isn't useful when the aim is to predict user behaviour from general features. ***log_type***, ***advertiser_id*** and ***anonymous_url_id*** were not used as they only consist of one unique record. All other features were initially included in the model training but through multiple experiments, it was found that the following features did not improve the model's accuracy:

1. *ip*
2. *city*
3. *region*
4. *user_tags*
5. *creative_id*
6. *key_page_url*

This may be because they are irrelevant in this context or we are yet to found their optimal use.

Finally, we used the following list of features during the training of the models:

1. *hour* - raw
2. *weekday* - raw
3. *url* - raw - one hot encoded
4. *domain* - raw - one hot encoded
5. *os* - custom - one hot encoded
6. *browser* - custom - one hot encoded
7. *tags* - custom - one hot encoded
8. *ad_slot_id* - raw - one hot encoded

9. *ad_slot_width* - raw - one hot encoded
10. *ad_slot_height* - raw - one hot encoded
11. *ad_slot_format* - raw - one hot encoded
12. *ad_slot_exchange* - raw - one hot encoded
13. *ad_slot_visibility* - raw - one hot encoded

Further, we experimented with three feature engineering techniques: combination features, feature hashing and one hot encoding while we also built a number of custom features.

6.2 Combination Features

A combination feature is a single feature formed using two single features. For example, a combination feature *ad_slot_width_height* is formed by adding *ad_slot_width* and *ad_slot_height* together. We experimented with multiple combination features, however we found that it did not improve the model's accuracy.

6.3 Feature Hashing

Feature Hashing is a technique which efficiently vectorizes features while applying dimensionality reduction and expansion. When features are hashed, they are compressed into a single sparse vector representation. Therefore, the output of the feature hasher is a single column. We experimented with this technique, but found that it didn't bring any improvements to the model's accuracy.

6.4 One Hot Encoding

One Hot Encoding is the process of encoding categorical features using a one-of-K encoding scheme. Let's consider an example where we have three categorical features:

- *hour* = {23, 18, 0}
- *weekday* = {3, 6, 4}
- *user_agent* = {ios_other, android_chrome, android_safari}

Based on several combinations of these three features, a user will click on an advert. In this example, one hot encoding blows up the feature space to three features which will get their own weights. During the one hot encoding process, every value in a feature is allocated a column in the output matrix. If a specific feature combination contains a value, it is represented by a 1 and positioned in the output matrix accordingly.

GraphLab Create [1] allows to specify "the maximum number of categories (per feature column) to use in the encoding" [2], through a parameter called *max_categories*. We found that setting the *max_categories* parameter to 120 resulted in the best AUC value.

6.5 Other

User Agent

The iPinYou data set stores the *user_agent* feature of the web browser used by the user who saw the impression. It follows the pattern {platform}_{browser}, so we split the feature into two separate features in order to differentiate between the platform and web browser where an impression appeared.

User Tags

The *user_tags* feature contains all related tags (separated with a comma) to an impression, so we create an encoded dictionary of tags in order to categorise impressions based on these labels. We called the resulting feature *tags*. Finally, this feature was added to the the set of features that were one hot encoded.

Ad slot floor price

The data set provided consists of a column holding the floor price of a specific ad slot. The values in this column range from 0 to 100+. In order to generalise this column's values, we experimented with the notion of "price buckets" as defined by Zhang et. al [3]. We defined 5 price buckets: 0, [1,10], [11,50], [51,100] and [101, $+\infty$] and assigned each value to its corresponding price bucket. This allows the prediction model to learn common factors between different historical ad slot floor prices from the past and improve its accuracy. However, in our experiments with this feature, the model's accuracy did not improve.

7 Forecasting Models

7.1 Logistic Regression

Submission path: *ctr-prediction/log_reg/*

Logistic Regression is a model that is used intesively for classification tasks where one's aim is to predict which category a new observation belongs to based on a training set which consists of observations whose categories are known. An example would be predicting whether an email is spam or non-spam. Logistic Regression employs a logistic function of a linear combination of features in order to predict whether a binary target is True or False.

Given a set of features, the model understands the probability that the label belongs to a class in the form of a logistic function of a linear combination of the features given.

Table 17 shows the notable hyperparameters that were trained in order to get an accurate Logistic Regression model.

Table 17: Notable hyperparameters of the Logistic Regression Model.

L1 Penalty	L2 Penalty	Convergence Threshold	Step size	Max iterations
0.0	0.01	0.01	1.0	20

7.2 Boosted Tree Regression

Submission path: *ctr-prediction/experiments/*

The Boosted Trees Model, or Gradient Boosted Machine (GBM) is a type of additive model that makes predictions by combining decisions from a sequence of base models and using a particular model ensembling technique called gradient boosting. Each base classifier is a simple decision tree using a subsample of the data. Table 18 shows the notable hyperparameters that were trained in order to get an accurate GBM model.

Table 18: Hyperparameters of the GBM Model.

Step size	Max depth	Column / row subsample	Max iterations
0.5	8	1.0 / 0.9	10

7.3 Support Vector Machine

Submission path: *ctr-prediction/experiments/*

Given a set of training examples, an SVM training algorithm builds a model that assigns new examples into one category, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped in such a way that the examples of the separate categories are divided by a clear gap that is as wide as possible (the decision boundary in maximized). Additionally, regularization (or penalty) and number of iterations are the only required model hyperparameters that need to be trained. As this model is non-probabilistic by nature, we implemented the Platt scoring algorithm [4] in order to be able to obtain output predictions as probabilities ([0-1]).

7.4 Field-aware Factorization Machines

Submission path: *ctr-prediction/libffm/*

Field-aware Factorization Machines are a recent variant of Factorization Machines, a form of factorisation models with feature engineering. It enables multiple categorical features to be taken into consideration in potentially high dimensional variables. Factorisation models can be learned in linear time whilst using Stochastic Gradient Descent. Field Aware FM learns a different relationship between every pair of features in its parameter space whereas basic FM uses a shared space.

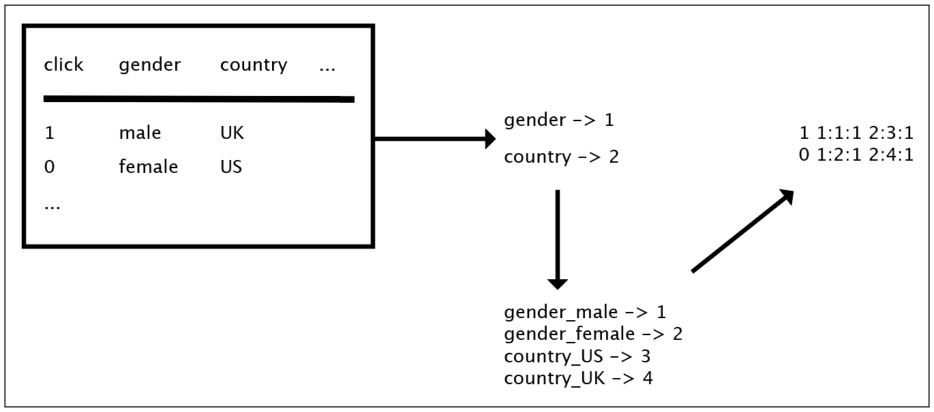

Figure 1: Data format for libffm

The implementation provides a python interface to execute the model, loads the data into a Dato SFrame and then passes the data structure into a c++ executable, containing the model and the parameters. Unfortunately the model is known for overfitting and the results generated were not either not usable or less than that predicted by logistic regression.

8 Evaluation

8.1 Logistic Regression

Before starting the training process of the Logistic Regression model, we first executed a hyperparameter search over ten different model configurations with the objective to get the best parameter values for the underlying data set. Table 19 presents the results obtained during this process. Further, Table 20 presents the iterative training progress of the Logistic Regression model, while Table 21 presents the model's Precision, Recall, F1 score, Log Loss and AUC evaluation metrics.

Table 22 presents an excerpt of the ROC Curve data of the Logistic Regression Model, while Figure 2 showcases the ROC Curve of the Logistic Regression Model in comparison with the ROC Curve of the Boosted Gradient Machine, our two best models. The Receiver operating characteristic (ROC) Curve consists of the True and False positive rates which are used to calculate the area under the curve (AUC).

GraphLab Create [1], the machine learning framework used in this project, consists of a parameter called *class_weights*. By default, this parameter is set to None, which means the model will only attempt to predict 0 labels as shown in Table 23. When set to "auto", the class weight is set "to be inversely proportional to number of examples in the training data with the given class" [5]. In other words, the model will begin to also attempt predicting 1 labels besides 0 labels as shown in Table 24.

Table 19: Model Hyperparameter Search for the Logistic Regression Model.

Model ID	L1 Penalty	L2 Penalty	Target	Training Acc.	Validation Acc.
9	100.0	0.0	click	0.999270694483	0.999273913501
8	100.0	0.0	click	0.999270694483	0.999273913501
1	0.1	0.001	click	0.999270694483	0.999273913501
0	0.001	10.0	click	0.999270694483	0.999273913501
3	0.0	1.0	click	0.99934951723	0.999179206566
2	1.0	0.01	click	0.999270694483	0.999273913501
5	0.001	0.01	click	0.999270694483	0.999273913501
4	100.0	0.1	click	0.999270694483	0.999273913501
7	0.1	10.0	click	0.999270694483	0.999273913501
6	100.0	10.0	click	0.999270694483	0.999273913501

Table 20: Logistic Regression Model iterative training progress.

Iterations	Passes	Elapsed Time	Training Acc.	Validation Acc.
1	2	8.100298	0.686048	0.685994
2	3	12.396830	0.693924	0.693991
3	4	16.601799	0.696544	0.696334
4	5	20.944276	0.697035	0.696948
5	6	25.157169	0.697062	0.696962
6	7	29.742889	0.697046	0.696959
7	8	34.197853	0.697046	0.696962
8	9	38.582071	0.697046	0.696959
9	10	42.839751	0.697047	0.696959

Table 21: Logistic Regression Model evaluation metrics.

Metric	Score
Precision	0.0017458867601660326
Recall	0.7294685990338164
F1 Score	0.00348343637538064
Log Loss	0.5184005159729493
AUC	0.7961598897596291

Table 22: Excerpt of the ROC Curve of the Logistic Regression Model.

Threshold	FPR	TPR	P	N
0.0	1.0	1.0	207	284883
1e-05	0.987545764402	1.0	207	284883
2e-05	0.982515629223	1.0	207	284883
3e-05	0.967042610475	1.0	207	284883
4e-05	0.947652194059	1.0	207	284883
5e-05	0.926878753734	1.0	207	284883
6e-05	0.898983091304	0.995169082126	207	284883
7e-05	0.882878234222	0.990338164251	207	284883
8e-05	0.871877226791	0.990338164251	207	284883
9e-05	0.860995566601	0.985507246377	207	284883

Table 23: Logistic Regression Model confusion matrix with *class_weights* parameter set to None.

Target Label	Predicted Label	Count
1	0	207
0	0	284883

Table 24: Logistic Regression Model confusion matrix with *class_weights* parameter set to ”auto”.

Target Label	Predicted Label	Count
1	0	56
1	1	151
0	0	198545
0	1	86338

8.2 Boosted Tree Regression

Before starting the training process of the GBM model, we first executed a hyperparameter search over ten different model configurations with the objective to get the best parameter values for the underlying data set. Table 25 presents the results obtained during this process, where Model ID 9 was finally used for training, which obtained the predictions presented in Table 26. The *class_weights* parameter was also set to ”auto” in this model.

Table 25: Model Hyperparameter Search for GBM.

Model Id	Column Subsample	Max. Depth	Max. Iterations	Row Subsample
9	1.0	8	10	0.9
8	0.8	8	10	1.0
1	1.0	6	100	1.0
0	0.9	4	100	1.0
3	1.0	6	100	0.9
2	1.0	6	10	0.9
5	0.9	6	100	0.9
4	0.9	6	50	0.9
7	0.9	6	10	0.9
6	0.9	4	50	1.0

Table 26: GBM confusion matrix.

Target Label	Predicted Label	Count
1	0	65
1	1	142
0	1	79665
0	0	205218

8.3 Support Vector Machine

We also tried implementing an SVM model to our RTB dataset, however results didn’t improve our previous models. Table 27 presents the performed hyperparameter tuning while table 28 shows the summary of predictions on the validation set.

Table 27: Model Hyperparameter Search for SVM.

Model ID	Penalty
9	1.0
8	10.0
1	0.001
0	0.01
3	0.001
2	1.0
5	1.0
4	0.1
7	0.1
6	10.0

Table 28: SVM confusion matrix.

Target Label	Predicted Label	Count
1	0	58
1	1	149
0	1	82792
0	0	202091

9 Model Comparison and Conclusion

Figure 2 presents the ROC Curve and Table 29 presents a model comparison summary of our two best models: Logistic Regression and Gradient Boosted Machine.

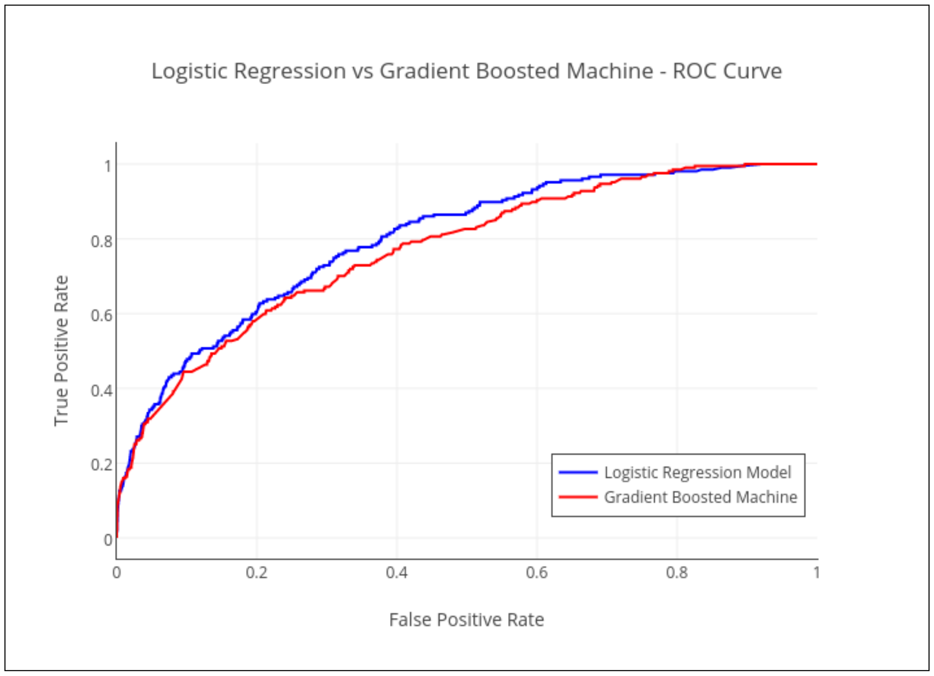

Figure 2: ROC Curve of our two best models: Logistic Regression and GBM.

Even if the GBM model obtains AUC values close to the Logistic Regression model, the former is prone to over fitting. On the other hand, the SVM model could not overcome the predictions score obtained with the first two models, but it is possible that a different set of features would boost its performance but once again, with the risk of a possible over fitting event. Our best score in the Kaggle competition is an AUC value of 0.80064 calculated against 50% of the test set.

Table 29: Model comparison summary.

Model	**AUC**	**Recall**
Logistic Regression	0.7961598	0.7294685
Boosted Gradient Machine	0.7718869	0.6618357
Support Vector Machine	0.6100004	0.7198068

References

[1] Dato: GraphLab Create. *Sophisticated machine learning as easy as "Hello, World!"*. URL: `https://dato.com/products/create/` (visited on 12/04/2016).

[2] GraphLab Create: One Hot Encoder. *GraphLab Create API: One Hot Encoder*. URL: `https://dato.com/products/create/docs/generated/graphlab.toolkits.feature_engineering._one_hot_encoder.OneHotEncoder.html` (visited on 12/04/2016).

[3] Weinan Zhang et al. "Real-time bidding benchmarking with iPinYou dataset". In: *arXiv preprint arXiv:1407.7073* (2014).

[4] John Platt et al. "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods". In: *Advances in large margin classifiers* 10.3 (1999), pp. 61–74.

[5] GraphLab Create: Logistic Classifier. *GraphLab Create API: Logistic Classifier*. URL: `https://dato.com/products/create/docs/generated/graphlab.logistic_classifier.create.html#graphlab.logistic_classifier.create` (visited on 12/04/2016).