Click Through Rate Prediction

Robert Espinoza
Molly Zhang

CMPS242

December 19, 2014

1 Abstract

This project is on click through rate prediction for web advertising in a big data setting, more specifically predicting whether a particular user will click on a web advertisement using a large n-dimensional training dataset. The methodology prescribed for this particular problem involves testing a selection of classification learning algorithms with the provided data, and then deciding which models perform better. We will take two different approaches to solve the problem - naive Bayes and logisitc regression. Each one of the algorithms will involve feature selection as well as tuning specific parameters, and will be compared against each other to determine the best performing learning model for the problem described. Our best performance result is an efficient L1 and L2 regularized logistic regression, using a hash-trick to improve scalability and reduce memory usage of the large dataset in this project.

2 Introduction

Web advertising has rapidly been growing over the years as mobile technology and the Internet have become accessible to a larger population. Click through rate is a commonly used metric in the online advertising community which represents the percentage of clicks. Predicting whether a mobile advertisement will be clicked on by a user is a binary classification problem that allows machine learning algorithms to excel. One of the main issues to highlight with click through rate data is high dimensionality and large number of examples. For instance, the features are usually categorical and if one is to one-hot-encoding all the categorical features, the number of features can rapidly rise up to more than 1 millions based on the number of categories available in each feature. In addition the number of training examples can reach to the tens of thousands of millions. Both issues have to be addressed so that the problem would not be computationally prohibitive.

The goal of this research is to successfully predict if a user will click on a given advertisement which is essentially a probability, $\mathrm{P}\,(Y = click|x)$ where Y is a binary random variable, and x is the set of features. The approach taken is to test a set of different classification algorithms, specifically naive Bayes and logisitc regression. Logistic regression is a popular example of discriminative classifiers, where naive Bayes is an example of a generative classifier. According to Ng and Jordan's paper regarding a comparison of discriminative versus generative classifiers:

> Generative classifiers learn a model of the joint probability,$\mathrm{p}\,(x, y)$, of the inputs x and the label y, and make their predictions using Bayes rule to calculate $\mathrm{p}\,(x|y)$, and then picking the most likely label y. Discriminative classifiers model the posterior $\mathrm{p}\,(\mathrm{y}|\mathrm{x})$ directly, or learn a direct map from inputs x to the class labels. [5]

As mentioned in the description of generative vs discriminative classifiers naive Bayes is computing the probability $\mathrm{p}\,(x|y)$, while logistic regression is computing the probability $\mathrm{p}\,(\mathrm{y}|\mathrm{x})$. More specifically a two class logistic regression can be defined as the posterior probability of class $\mathcal{C}_1$ by applying the logistic sigmoid function (1.1) to the linear function $\mathbf{w}^T\mathbf{x}$, where $\mathbf{w}$ is the weight vector and $\mathbf{x}$ is the feature vector.

$$\sigma\,(a) = \frac{1}{1 + exp\,(a)}$$

$$\mathrm{p}\,(\mathcal{C}_1|\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x}\right)$$

In order to determine the parameters of the logistic regression the likelihood function must be derived, which will be used to find the maximum likelihood function. In the case of the logistic regression the derivative of the logistic sigmoid function is required for the computation of the maximum likelihood function. The likelihood function is transformed by the negative log, which allows the gradient to easily be found. The negative logarithm of the likelihood also represents the cross entropy error function. The gradient of the error function is then set to zero in order to find the logistic regression parameters. When the parameters are determined the posterior probability distribution $\mathrm{p}\,(\mathcal{C}_1|\mathbf{x})$ is computed. Improvements in the error minimization can be made by including a regularization term to the optimization function. By including the L1 or L2 regularization term sparsity of the weight vector can be achieved.

On the other hand, naive Bayes has a different approach in estimating the probabilistic outcomes of the model. As discussed when describing the generative versus discriminative classifiers the naive Bayes model is computing the posterior probability using Bayes' rule, which states

$$\mathrm{p}\,(\mathcal{C}_1|\mathbf{x}) = \frac{\mathrm{p}\,(\mathcal{C}_1)\,\mathrm{p}\,(\mathbf{x}|\mathcal{C}_1)}{\mathrm{p}\,(\mathbf{x})}$$

The denominator can be avoided from the computation, which gives the following proportion,

$$\mathrm{p}\,(\mathcal{C}_1|\mathbf{x}) \propto \mathrm{p}\,(\mathcal{C}_1)\,\mathrm{p}\,(\mathbf{x}|\mathcal{C}_1)$$

The model is called naive Bayes because the model makes the assumption that each of the $\mathbf{x}_i$ are conditionally independent from each other, which simplifies the probability distribution above. When the conditional independence assumption is made the model is simplified to,

$$\mathrm{p}\left(\mathcal{C}_1|\mathbf{x}\right) = \mathrm{p}\left(\mathcal{C}_1\right)\prod_{i=1}^{n}\mathrm{p}\left(\mathbf{x}_i|\mathcal{C}_1\right)$$

By using the RHS of the above formula, the posterior probability distribution can be computed. The classifier is constructed using the posterior probability from using Bayes' rule for each class given a particular observation, and choosing the maximum probability. The optimization technique described is MAP(maximum a posterior probability), which is used to define the decision rule of the classifier. A Laplace smoothing parameter may be used to prevent zero probabilities, and improve the performance of the model. The models are evaluated using log loss function and ROC curve in order to determine the performance of the models. These are metrics to evaluate performance of a classifier. The log loss specifically evaluates the error of the probabilities generated against the observed labels. The log loss function is defined as

$$LogLoss = \frac{1}{n}\sum_{i=1}^{n}\left[y_i log\left(\hat{y}_i\right) + \left(1-\hat{y}_i\right) log\left(1-\hat{y}_i\right)\right]$$

The ROC curve is a graph used to determine what decision threshold is optimal for the posterior probabilities. The axis of the ROC curve are the true positive rate versus the false positive rate. By using these metrics it was determined that the logistic regression classifier outperformed the naive Bayes classifer.

The data being used to train the classifier was generated by Avazu, and the format of this project is a kaggle competition [2]. Avazu is an advertising platform that specializes in cross-device advertising and mobile game publishing. There is a total of 23 features and 1 response column in the dataset. The response column is clicks, which is a binary variable. The 23 features in the data are listed:

1. ID
2. Hour
3. Banner Position
4. Site ID
5. Site Domain
6. Site Category
7. App ID
8. App Domain
9. App Category
10. Device ID
11. Device IP
12. Device Model
13. Device Type
14. Device Connection Type
15. C1, C14-C24

Item 15 in the feature list above includes the anonymous features of the data, where the definition of the features C1, C14-C24 are undetermined. The dataset has approximately 40 million observations. The interesting and difficult caveat

regarding the data is that the features are entirely categorical values, and most of the features have a large number of possible values the feature vector can take, which slows down computation.

The problem of classifying click through rates stands to be an interesting question because utilizing the features that were collected from actual users, algorithms can predict whether a user will click or not click o an advertisement. The goal is to find an algorithm that can be as accurate as possible. This may stand to be a difficult task since the data is not evenly disbursed between click and not click observations, where click observations are less common. Essentially by looking at a users characteristics and habits, a model can determine user's actions. The additional importance of the problem is how to deal with a data of high dimensionality, and applying scalable solutions to data of that nature. The problem was first approached by running preliminary models on R, using the glm function with the family=binomial parameter set from the glm package to train a logistic regression model. The nature of the data does not allow us to collect sufficient amount of data using the glm() in R, so only a a preliminary model was ran, and the FTRL-Proxima approach to logistic regression was favored to collect experiental data. The naive Bayes model was trained using the naiveBayes() function in the e1071 package. The third model that was tested was a more sophisticated version of logistic regression using a hashing method. The algorithm used is the FTRL-Proximal(follow the regularized leader - proximal) executed on Python, which provided improved results from the preliminary logistic regression and naive Bayes results generated from R. The goal was to see if a generative or discriminative classifier would perform better, and according to the results from our experiment data the logistic regression classifier outperformed the naive Bayes classifier. Therefore our results indicate that discriminative classifiers outperformed generative classifiers for the Avazu click though rate data.

3 Related Work

Previous research in the area of click through rate prediction modeling has been worked on before. Advertisement is one of Google main revenue contributors, and has done extensive research on advertising data and modeling. One paper published by Google that is used in this research is "Ad Click Prediction: a View from the Trenches", which outlines a particular model to handle large datasets with logistic regression, since Google analyzes online data that must be continuously updated. The paper discusses the FTRL-Proximal online learning algorithm, which excels in implementing sparsity and convergence properties, and is scalable for large datasets. FTRL-Proximal is an efficient way to implement stochastic gradient decent, which reduces the amount of memory being used. The approach in the paper also includes a both L1 and L2 regularization for sparsity reasons. The algorithm is compared to Regularized Dual Averaging (RDA), FOBOS, and Online Gradient Descent- Count (OGD-Count).

Another paper to highlight is "Simple and scalable response prediction for

display advertising. The data used in this research paper are live traffic logs from Yahoo!'s Right Media Exchange, which is on of the largest ad exchanges. The goal of the study was to implement scalable models to large datasets, in this paper a hashing trick for logistic regression is discussed, which helps tackle the issue of the dummy coding required to utilize logistic regression on categorical variables in the data, where the f^{th} feature has c values, and requires c binary features for the f^{th} feature. This is required for each categorical feature in the data, this causes the dimensionality to explode very quickly.

> If we have **F** features and the f^{th} can take c_f values, this encoding will lead to a dimensionality of
>
> $$d = \sum_{f=1}^{F}$$
>
> (McMahan, Holt, et al. A:7)

The hashing trick discussed in the paper helps deal the dimensionality of the problem, especially since online advertising data usually has features with a large value c.The same paper also mentions the effects of subsampling a data set of this nature. There are two ways to subsample a dataset similar to what we will be working with. One is to keep all the positive examples, and subsample the negative examples. This technique may be beneficial since the occurrence of positive examples has a lot lower percentage in comparison to the negative examples in the data. The second technique to subsample is overall subsampling, which samples from the entire data set. The effects of subsampling is a trade-off between computational complexity and accuracy of the model. Another related work pertaining to this particular problem, is "On Discriminative vs Generative classifiers: A comparison of logistic regression and naive Bayes", which discusses the advantages of choosing logistic regression over naive Bayes, when common opinion assumes naive Bayes assumes the contrary. The paper makes a formal analysis of the algorithms, and provides experimental data. The paper shows the following proposition:

> Let h_{Gen} and h_{Dis} be any generative-discriminative pair of classifiers, and $h_{Gen,\infty}$ and $h_{Dis,\infty}$ be their asymptotic/population versions. Then $\varepsilon(h_{Dis,\infty}) \leq \varepsilon(h_{Gen,\infty})$.

From the above proposition a logistic regression classifier should exhibit lower asymptotic error compared to naive Bayes. This becomes an interesting claim because a lower error rate essentially means a high accuracy rate for the model, which is ideal. Since the problem and dataset has originated from Kaggle.com, the Kaggle community has contributed to the progress of the problem, as well as provided further insight to the problem.

4 Data

there are 24 columns and 40428968 rows in the training data, which is 5.9GB in size. There are 23 columns and 457465 rows in the test data, which is 673MB in size. The following text is a column by column description/histogram of each feature:
id: there are 40428967 ids, meaning each row has a uniq ID, therefore the in the trainig script "id" column is removed.
click: there are 33563901 rows that are "1" in click and 6865066 rows that are "0" in click, the corresponding percentage of click or not-click is 83.02% and 16.98%. This feature is not included in the test data, for obvious reason.
hour: made of 8 numbers corresponding to YYMMDDHH, since YY (year) and MM (months) are the same across all training data, information in YYMM is discarded. after discarding YYMM, the hour column made of DDHH is separated furtur into three more columns: dd (date), hh (hour) and dw (day of week). The following text describes the statistics of each of the three sub-features:
dd: the date of the data ranges from 21st to 30th, each has from 3.2 million to 5.3 million rows
hh: the hour of the data ranges from 0 to 23, each has 0.8 million to 6.0 million rows
dw: the day of week data ranges from a number 0 to 6, each has from 3.2 million to 9.4 million rows
banner pos: there are 7 possible banner positions, with about 72.0% of the banners at position 0 and 27.8% of the banners at position 1, the rest of the 5 positions occur relatively rarely
site id: there are total 4737 sites ids
site domain: there are total 7745 site domains
site category: there are a total of 26 site categories
app id: there are 8552 different app ids
app domain: there are 559 different app domains
app category: there are 26 different app categories
device id: there are a total of 2686408 device ids
device ip: there are a total of 6729487 device ips
device model: there are a total of 8251 different device models
device type: there are 5 different device types
device conn type: there are 4 different device connection type
C1: there are 7 different C1 categories, each category has from 5000 rows to about 3.7 million rows
C14: there are a total of 2626 C14 feature categories
C15: there are 8 C15 feature categories, each has from 1621 rows to 37.7 million rows
C16: there are 9 C17 feature categories, each has from 1621 rows to 38.1 rows
C17: there are a total of 435 C17 feature categories
C18: there are 4 different C18 features, each has from 2.7 million to 16.9 million rows

C19: there are a total of 68 C19 feature categories
C20: there are a total of 172 C20 feature categories
C21: there are a total of 60 C21 feature categories

if one is to one-hot-encode all the categories in each feature, there are a total of 9449237 (9.4 million!) features. In order to process data in a computationally feasible manner, two tricks are used to minimize computational time and resources, with the help of kaggle competition forum discussion [2]. The combination of the two tricks allows for a reduction of memory usage from tens of hundreds of GB to a few MB.
first trick: instead of saving all the data in the memory, we used a python function called generator, which is an iterator that will iterate through the data row by row, without saving all data in memory. Instead, only one row at a time is saved in the memory.
second trick: out of all the 9.4 million one-hot-encoded features, for each example x, the vast majority of x features are 0s, with very few occurrences of 1s. The feature hashing takes the mode of x[key+value]%D and append each hash to x, which allows for the elimination/collapsing of all features that are of 0 values. This way the difficulty of compution and requirement of memory is vastly reduced.

5 Methodology

The procedure and methodology to analyze the data is as follows. First data processing was done in python, preliminary results were collected in R using glm() from the glm package and naiveBayes() from the e1071 package in order to model logistic regression and naive Bayes classifiers respectively. At the same time, we tested the FTRL-Proximal approach with L1 and L2 regularization for the logistic regression in python. Cross-validation was used for each of the approaches to test the different hyper parameters, and feature selections.

The data required pre-processing to parse out the required information from the 'hour' column in the data. The 'hour' column in the data is initially in the format YYMMDDHH, since the data is only from a span of 10 days, the YY and MM are irrelevant. Therefore from the column, three additional features were extracted and the initial 'hour' column was discarded. The three additional features extracted are date of month, hour of day, and the day of the week. Another irrelevant feature in the dataset was the 'ID' column, which are user ID's that are not necessary in order to make predictions. The entire dataset was also subsampled in order to effectively run the data in a timely manner. A two percent random subsample was taken from the dataset, and when the percentage of clicks were checked in the new subsampled data it was not significantly different from the original dataset. However, training with only 2% data yields very poor result in turns of paramater tuning, mostly because of the under-representativeness of the data. Therefore the result is not reported.

The logistic regression classifier on R was not scalable with the size of the data, even when using only two percent subsampled data. The issue with using

the glm() method was the dummy variable encoding. Since there are large values of c_f some reaching over 1000, then this becoming an major issue for packages such as glm. One model was tested with specifically selected features. The features selected for the logistoc regression classifier implemented by glm() are hour of day, day of week, app category, site category. Including any additional features into the glm method exceeded the memory limits.

The naive Bayes classifier was more successful being implemented in R on the two percent subampled data set, but could still not handle the entire set of features at once. The approach taken to test the naive Bayes model on the subsampled dataset was to select three different models using a different combination of features, then for each of the models testing different Laplace smoothing parameters. The three different models chosen are as follows. The features used for Model 1 are hour of day, day of week, banner position, site ID, site domain, site category, app domain, category device ID, device model, device type, and device connection type. The features used for Model 2 are hour of day, day of week, banner position, site ID, site domain, site category, app domain, app category, device model, device type, device connection type, C1, and C14-C21. The features used for Model 3 are day of month, hour of day, day of week, banner position, site ID, site domain, site category, app domain, app category, device ID, device model, device type, device, and device connection type. Model 1 excludes any of the anonymous features in the data, Model 2 includes all of the anonymous features in the data, and Model 3 has the same features as Model 1 with the addition of the day of the month. 39 different Laplace smoothing parameters were tested for each of the three different models. The Laplace smoothing parameters include [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, .9, 0.95, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20].

The third method used to model the data is the FTRL-Proximal approach for logistic regression in python. This was the method that performed the best in both time complexity and accuracy. In addition to implementing the FTRL-Proximal, a hashing method was used to reduce memory required to run the algorithm on the data. FTRL-Proximal is a form of an online learning algorithm, without any regularization terms FTRL-Proximal is exactly the same a Online Gradient Descent, where FTRL-Proximal performs the following updates on the weights,

$$\mathbf{w}_{t+1} = \underset{w}{\operatorname{argmin}} \left(\mathbf{g}_{1:t} \cdot \mathbf{w} + \frac{1}{2} \sum_{s=1}^{t} \sigma_s \|\mathbf{w} - \mathbf{w}_s\|_2^2 + \lambda_1 \|\mathbf{w}\|_1 \right)$$

where $g_t \in R^d$ is the t^{th} gradient, and $g_{1:t} = \sum_{s=1}^{t} g_s$. The per-coordinate learning rate is defined as

$$\eta_{t,i} = \frac{\alpha}{\beta + \sqrt{\sum_{s=1}^{t} g_{s,i}^2}}$$

where α and β are hyper parameters of the algorithm. The outline of the FTRL-Proximal is as follows. First initialize $z_i = 0$ and $n_i = 0$ $i \in 1, ..., d$ for $t = 1$ to

T, and for each for each feature vector $_t$ compute the following,

$$w_{t,i} = \begin{cases} 0 \\ -\left(\frac{\beta+\sqrt{\mathrm{n_i}}}{\alpha} + \lambda_2\right)^{-1} (z_i - sgn\,(z_i)\,\lambda_1) \end{cases}$$

,

where λ_1 and λ_2 are the regularization parameters. Then proceed by computing $p_t = \sigma\,(x_t \cdot \mathrm{w})$ by using the above $\mathrm{w}_{t,i}$, where $\sigma\,()$ is the sigmoid function. Then for all i compute and iterate the following,

$$g_i = (p_t - y_t)\,x_i$$

$$\sigma_i = \frac{1}{\alpha}\left(\sqrt{n_i + g_i^2} - \sqrt{n_i}\right)$$

$$z_i \leftarrow z_i + g_i - \sigma_i w_{t,i}$$

$$n_i \leftarrow n_i + g_i^2$$

The above is a general outline of the FTRL-Proximal with L1 and L2 regularization for logistic regression. In addition to the algorithm described above, the model also implemented python generator function to avoid saving the entire dataset in memory. The model also used hashing trick to deal with the issue of the dummy variable encoding (one-hot-encoding) of the required by the logistic regression for the categorical features in the data. The outline of the hashing trick is as follows, for values of the F features, $v_1, ..., v_F$, there exists a family of hash functions h_f, and there is a number of bind d. We then proceed with the following statements,

$$x_i \leftarrow 0, 1 \leq i \leq d$$

and for $f = 1...F$

$$i \leftarrow [h_f\,(v_f) \bmod (d)] + 1$$

$$x_i \leftarrow x_i + 1$$

which should then return a vector of $(x_i, ..., x_d)$

With the use of FTRL-Proximal with regularization and the hashing trick in our model, this allows for the uses of the complete data set, in a memory efficient and time complexity efficient manner without the loss of accuracy. Sparsity reduces the dimensionality of the data, which allows for better time complexity. Reducing time complexity and memory is an important aspect in modelling our problem since we are dealing with high dimensional data, and since we lose a accuracy by subsampling the data. Subsampling was attempted but our results, as discussed later were not as accurate when compared to the accuracy of using the entire dataset.

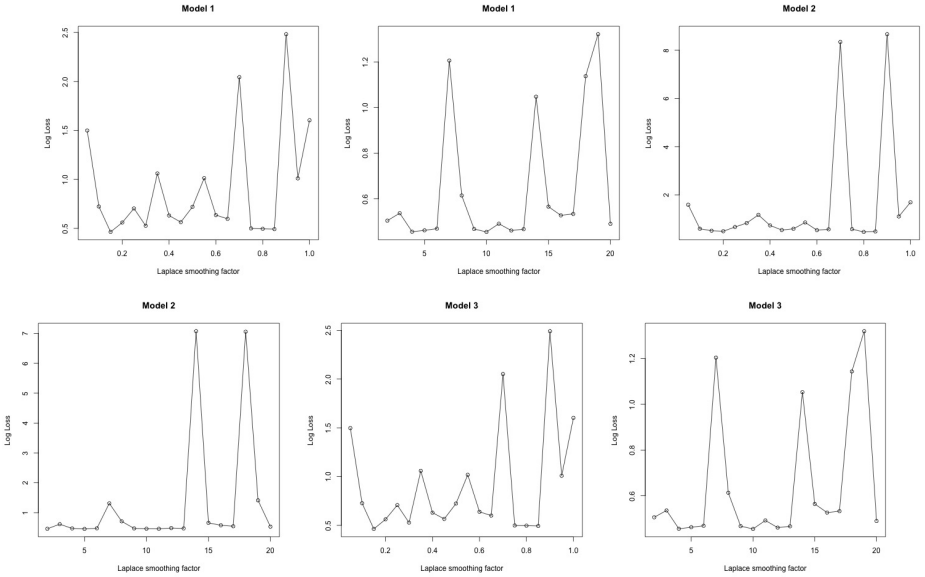
After analyzing the models using the different approaches described. The model with the minimum Log Loss value was chosen. Using the selected model,

an ROC curve was generated to select the optimal decision threshold. All of the models were trained, cross validated, then test whith independent datasets. The models implemented in glm() and naiveBayes used 70% of the subsampled data for training, 15% of the subsampled data for cross-validation, and another 15% of the subsampled dataset for testing. The FTRL-Proximal was tested using hold out cross validation method, where every 10th observation from the dataset was put into the cross validation set. Then it was tested using the kaggle test dataset.

6 Experimental Results

Results from R:

The results generated from running the subsampled data in R with the glm() logistic regression classifier yielded a log loss value of 2.09431, at a less than optimal running time. This model was discarded due to its poor performance. The naiveBayes() method was tested using Model 1, Model 2, and Model 3 on the subsampled data with varying Laplace smoothing parameters to see which parameter yielded the best log loss value. The following graphs shows the results for each of the models for the varying Laplace smoothing parameters on the test data set.



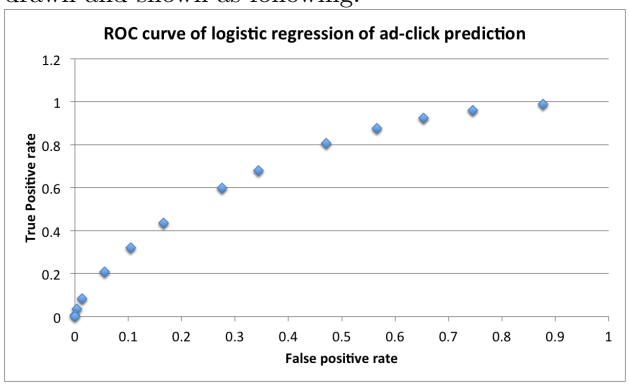
The minimum Log loss for each model is Model 1 yielded a value of 0.4547867, Model 2 yielded 0.4572809, Model 3 yielded a value of 0.4548537. All of these values are improvements from teh glm() logistic regression classifier. Model 1 yielded the minimum log loss value with a Laplace smoothing parameter of 10.

Results from FTRL proximal algorithm:

As described previously with FTRL-Proximal, the following parameters are tuned to achieve the minimum log loss value based on both 10% training data validation and the actual test data performance based on submission to kaggle.com. From the results of the experiments, the subsampled data did not perform as well as the complete dataset, which was expected. The log loss for the models trained on the subsampled data, which were then tested with the test submission performed significantly lower compared to the models trained on the complete dataset. By using holdout cross-validation technique to evaluate the trained model, the best yielded results for the parameters of the model are as follows:

alpha = 0.1, alpha is the learning rate
beta = 1, beta is the smoothing parameter for the adaptive learning rate
D = 2^{24}, D is the number of weights to use in the feature hashing. The lower D is, the less available numbers there are for hashing the features, the higher D is, the higher number of possible features there are to be hashed. However, the length of the program running time increase with D. Finding an optimal tradeoff between lower D thus faster program and higher D thus slower program results in the above value, at which point the increase of prediction accuracy is very marginal and is not worthy of the extra time spent on running the program.
L1 = 1, L1 regularization, the higher L1 is the more regularization.
L2 = 1, L2 regularization, the higher L2 is the more regularization.

The best log loss we achieved is 0.3945038, leading to a rank of 165 out of 866 participating teams as of the writing of this report. Based on these model parameters, an ROC curve with different thresholds ranging from 0.02 to 0.9 is drawn and shown as following:



By referring to the ROC curve allows us to determine an appropriate decision threshold for our model, based on the desired trade off between specificity and sensitivity. In the setting of our problem we may be more interested in a high true positive rate and be willing to sacrifice the cost of a false positive rate since the action of incorrectly classifying a click does not have high costs.

7 Conclusion

Logistic regression combined with FTRL-Proximal and feature hashing is the highlight of this report. The purely categorical nature of each feature and binary classification nature of this problem also implied logistic regression to be a good fit for this problem. For future endeavors, the following things could be done in order to achieve better prediction:
1. More thoughtful and innovative feature prepossessing, such as discarding feature categories that are of extremely low frequencies or those that generate more noise than merits. We noticed that there are two types of ad: mobile and web. Seperating the learning of the two types of ads might help to improve the learning accuracies as well.
2. Currently the program takes 40-50 minutes each run in order to train with 90% training examples, making parameter tuning difficult because of the time consumed. Implementing parallel computing methods to speed up the process would be highly desirable and a valuable assets to similar problems with large datasets.
3. One downside of the data provided in the kaggle competition is that, the company which provided the data would like to protect its business secret by anonymization of some features and sub-sampling it's original click-non-click data. The result of the feature anonymization is that it removes possible insights in which we as problem solvers could have picked up if we know what the data is. The adverse effect of the data sub-sampling is that the data could stop being correlated to the time and date when it was collected.
4. An additional curiosity for further research is the implementation of feature hashing technique to other learning algorithms such as SVM, neural networks and decision trees.
5. Thanks to the forum discussions and code sharing on kaggle.com, we learnt to address the memory/computational issues in python and was introduced to the FTRL-promixal algorithm. Without the help of the online community, we would have a much harder time finishing the project.

References

[1] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, Jeremy Kubica. "Ad Click Prediction: a View from the Trenches". KDD'13, August 11–14, 2013

[2] `https://www.kaggle.com/c/avazu-ctr-prediction`

[3] Bishop, Christopher. *Pattern Recognition and Machine Learning*. New York: Springer Science+Business Media, LLC, 2006. Textbook.

[4] Oliver Chapelle, Eren Manavoglu, Romer Rosales. "Simple and scalable response prediction for display advertising".

[5] Andrew Y. Ng and Michael I. Jordan. "On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes". 2006.