Machine Learning and Reinforcement Learning Applications in Real Time Bidding

Paul Pereira

April 22, 2017

1 Introduction

The ad business has become a multi-billion dollar industry. Many of the ad placements online are sold over marketplace organized as a second price auction. Many millions of these auctions can occur everyday. In order to participate in these auctions, it is therefore necessary to build an automated bidder capable of deciding, based on information provided about the ad placement and the user that will see the ad, how much to bid on an auction. This is usually done by using machine learning techniques to estimate the value of the ad placement for the advertiser and then by using this value to make a decision. In this report, I will first be looking at different methods used to estimate the value of the ad placement. I will then try to use Q-Learning to build a bidder that can outperform the more commonly used linear bidder.

2 CTR Prediction

The first step involved in creating a real-time bidding agent for the advertisement marketplace is to determine which KPI, or measure we will use to determine when our algorithm performs well. The most simple one to use is the click-through rate which can be seen as the probability that the ad will be clicked on given that some advertiser won the auction on that ad. In this section we will therefore look at different ways of completing this classification task.

2.1 Data Processing

The dataset that we use is obtained from the iPinYou global real-time bidding competition from 2013. The dataset provides information over 40M examples of auctions that occurred, including the features that were used at the time by the bidders to decide how much to bid on the ad as well as whether or not the ad was clicked on by the visitor. In this analysis, we decided to limit the dataset to a single advertiser/ad campaign. The purpose of doing so was to both reduce the size of the dataset as well as improve our prediction capabilities. The CTR of an ad placement depends

on the features provided by iPinYou, however it also depends on the quality and the content of the ad that is being shown to the user.

Once a single ad campaign was isolated. I removed any feature from the 30 provided that either could not be used to make a prediction (for example the price that was eventually payed for the ad since this is what we are trying to determine in the first place). In the end, this left me with 12 features, all categorical in nature. Table 1 provides a breakdown of those features as well as the number of categories for each features.

Once the features were extracted, I performed two kind of feature transformation to facilitate learning. The first was to use one-hot-encoding for every feature. This approach allows for no information to be lost during the transformation. For the second transformation, I first isolated every positive example in the dataset. Then, for every feature, for every class for that feature, I calculated the occurrence rate of that class among the positive examples. Finally, for the entire dataset, if example m had value c at feature j, then that value was replaced by the occurrence rate for value c of feature j. The main advantage of that method is that it keeps the numbers of features small and turns the categorical data into real values. This allows us to use decision trees as one of the classifiers. The disadvantage is that some information is lost as a result.

2.2 Logistic Regression and SVM

The first classifiers that I considered for this tasks were the classifiers we have learned about in class, namely Logistic Regression and SVM.

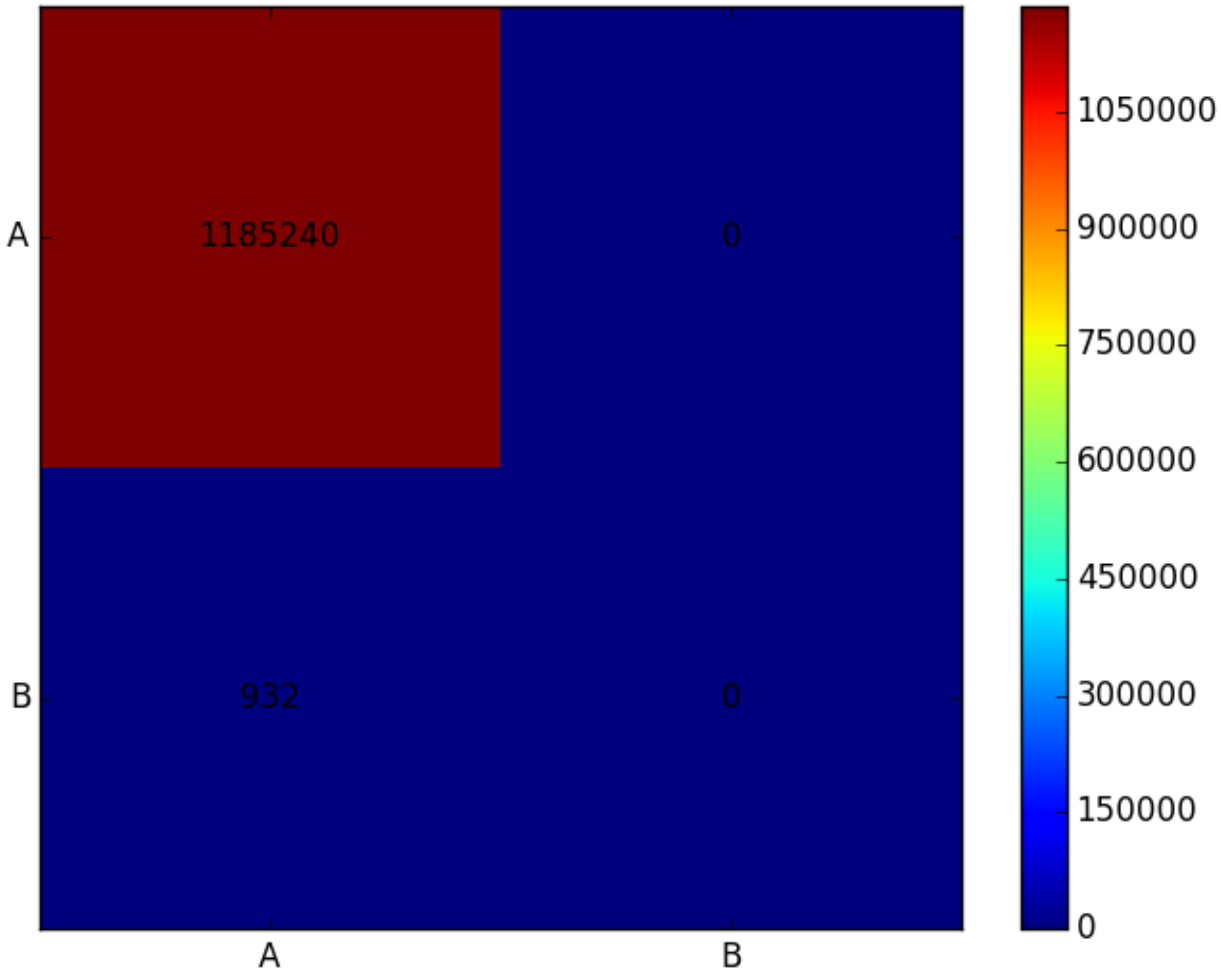
For these classifiers, I used the one-hot-encoding version of the dataset. Therefore, the number of features is very large (639986 features to be exact). Therefore I decided to see how Logistic Regression performed with and without using l1-regularization. L1-regularization can be used to drive the weights of some of the features to 0. Therefore, if by using L1-regularization I am able to perform similarly or better than without regularization, it could be an efficient way to reduce the dimensionality of the data.

In order to evaluate my models, I first split performed an 80-20 train-test split. I then used 10-fold cross-validation with the area under the ROC as the value to optimize. Initially I used the negative log loss function, however due to the class imbalance present in the data, it was a poor performance evaluator. The ROC curve plots the sensitivity of the data with respect to the specificity of the classifier. Therefore, if we have a class imbalance issue and all the examples are treated classified as negative the area under the curve should be small.Furthermore, I also looked at the confusion matrix once the parameters were selected and the model tested on the test set. From the cross-validation, I determined that using l1-regularization with regularization parameter of 1.2 was best. Table 2 shows the results of this procedure.

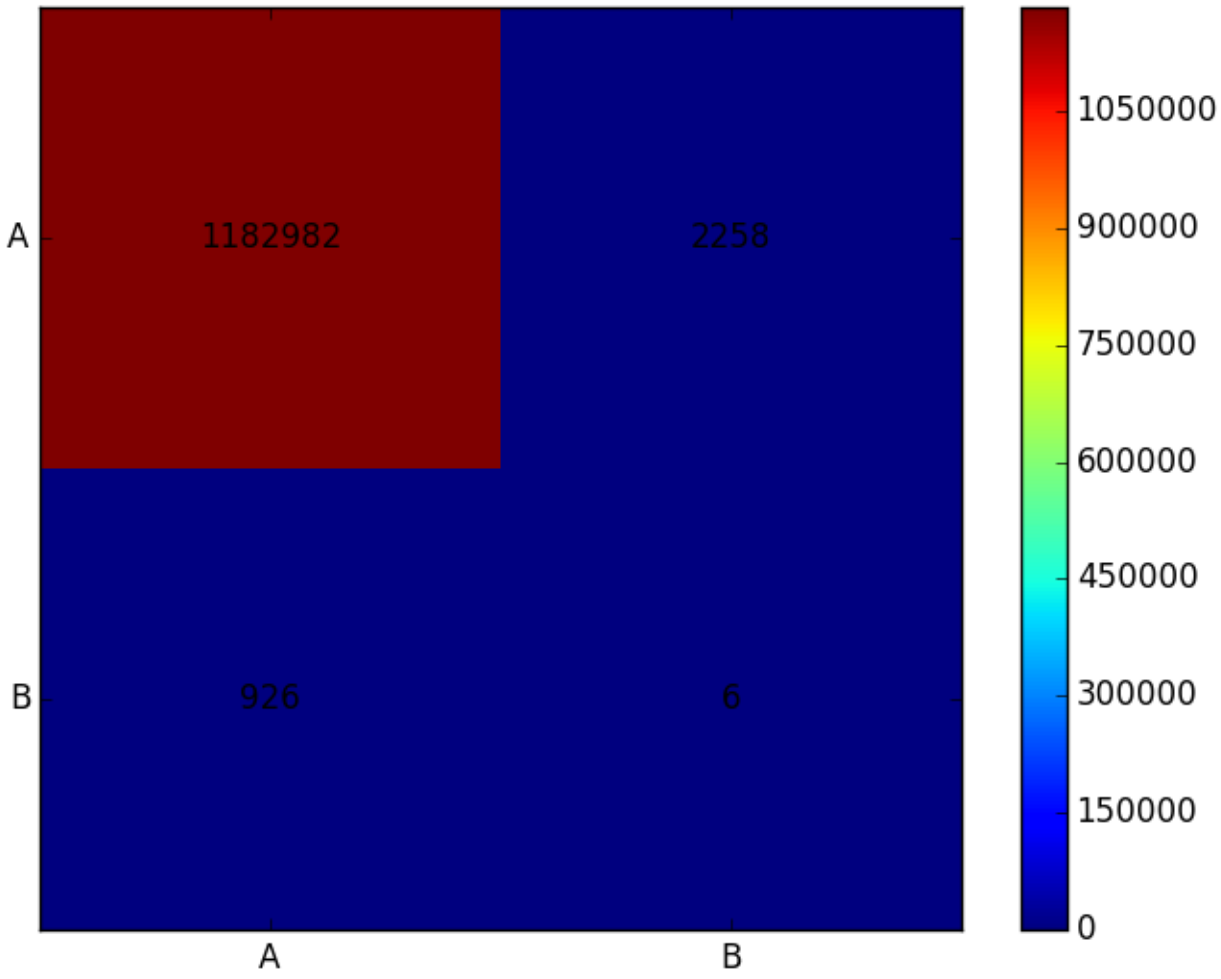
However, looking at the confusion matrix when testing Logistic Regression on the test set, we see that the results are poor and that due to the class imbalance, all the positive examples are misclassified as negative examples.

Table 1: AVG AUC / Reg Param

	No Reg	0.5	0.7	1	1.2
AVG AUC	0.711	0.715	0.712	0.74	0.76

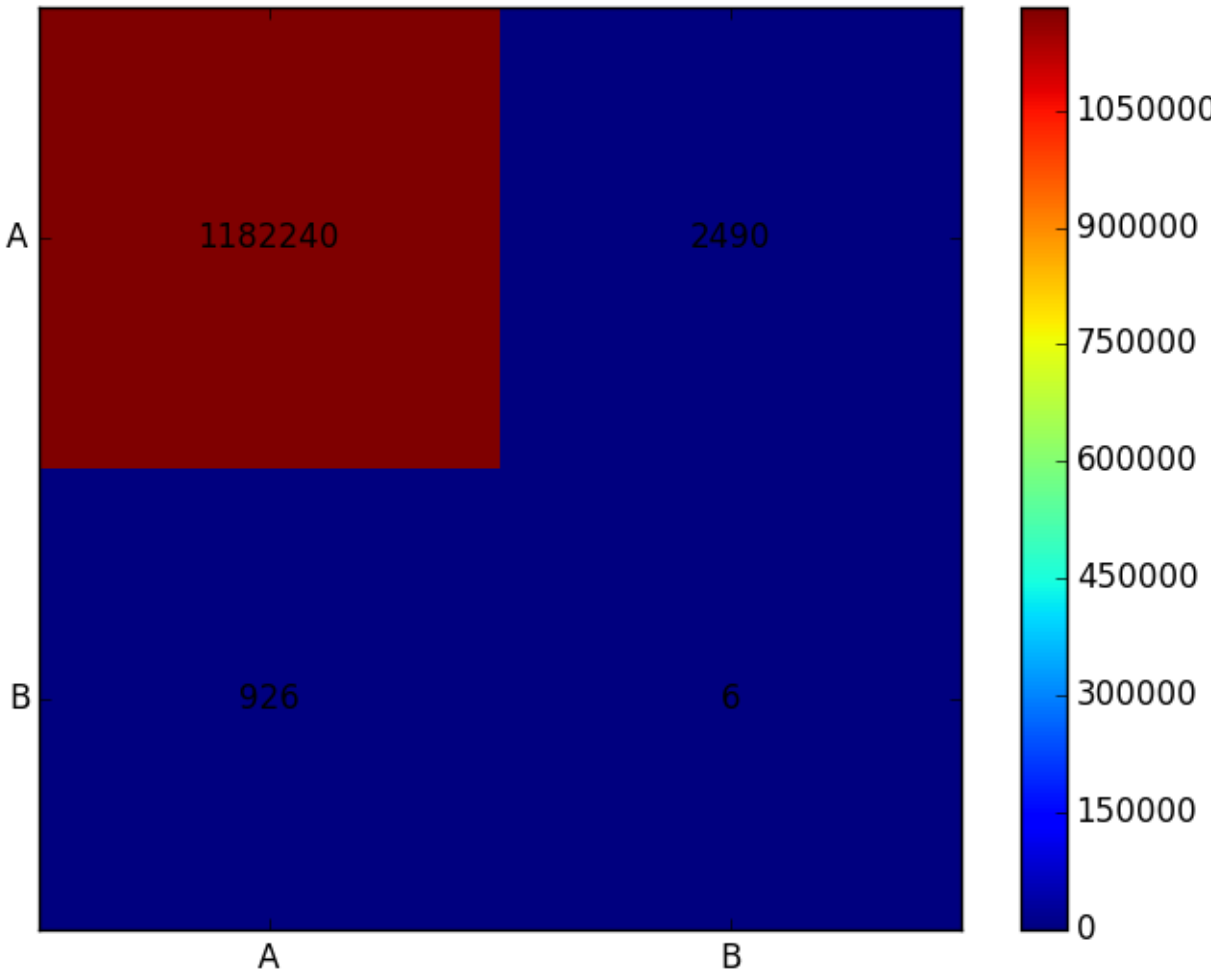


One attempt to deal with this issue was to use the class balance option of the scikit-learn package. This option modifies the impact that misclassifying a positive example has on the error function. The larger the class-imbalance ratio, the higher the weight of the impact. I kept the regularization parameter the same and tested this new model on the test set. As we can see from the confusion matrix, the results are different but not necessarily better.



The model does a better job at detecting positive examples, however the improvement is small (from 0 to 6) and the consequence is that now many negative examples are getting misclassified as positive.

Next I wanted to see whether or not an SVM would be able to deal with this problem in a better way. I opted to use the 'balanced' weight option on scikit-learn since it produced more interesting results in the case of linear regression. However, when I evaluated the SVM on my test set and generated the confusion matrix, it is apparent that the SVM performed no better than linear regression.



2.3 Boosted Decision Trees

The last classifier that we tested was the boosted decision trees classifier. A decision tree is a simple classifier that works well on real-values features. The main idea is to build a binary tree that splits the data into two subgroups at every node. Each node is parameterized by a feature it looks at and a threshold t that is used to seperate the examples. In order to decide which t to choose given a specific feature, one approach is to pick the t that maximizes the information gain:

$$t^* = argmax_t IG(X,t) = H(X|t) - H(X) \tag{1}$$

A single decision tree however is rarely enough to perform well on complicated classifications tasks. This is where ensemble methods come in. The idea of an ensemble classifier is to build many simple classifier: $h_1(x), h_2(x), ...h_n(x)$ and use a linear combination of the predictions from those

Table 2: 10 trees, changing depth

	1	5	10	100	500
AVG AUC	0.59	0.60	0.62	0.65	0.65

Table 3: depth of 1, changing number of trees

	1	5	10	100	500
AVG AUC	0.6	0.6	0.62	0.75	0.77

feature to perform the classification:

$$En(x) = \sum_{i=0}^{n} \lambda_i * h_i(x) \tag{2}$$

The reason that boosted decision trees are particularly well adapted to cases where class imbalance is an issue is that during the training of the boosted trees, the label of the examples are modified so as to put an emphasis on examples that have been misclassified so far. This means that if we have enough decision trees in our model, we have some of them that learn to recognize positive examples. One possible side-effect however is over-fitting. In order to ensure that our results are not due to over-fitting, I again split the data into 80-20 train-test. Then I used 10 fold cross-validation to try different parameters for the ensemble. There are two parameters we can play with. The first one is the number of boosted trees in the ensemble, the second one is the depth of each tree in the ensemble. Increasing the depth of the tree increases its predictive capabilities, but also the variance of the model. Increasing the number of trees in the ensemble also increases the variance of the model. Based on the cross-validation and once again using the auc metric, I determined that it was better to use a large number of trees rather than a a small number of trees of large depth.

However, when testing the ensemble on the test set, I once again found the results to be very poor. We are once again in a scenario where a few positive examples are being found but at the cost of many false positives.

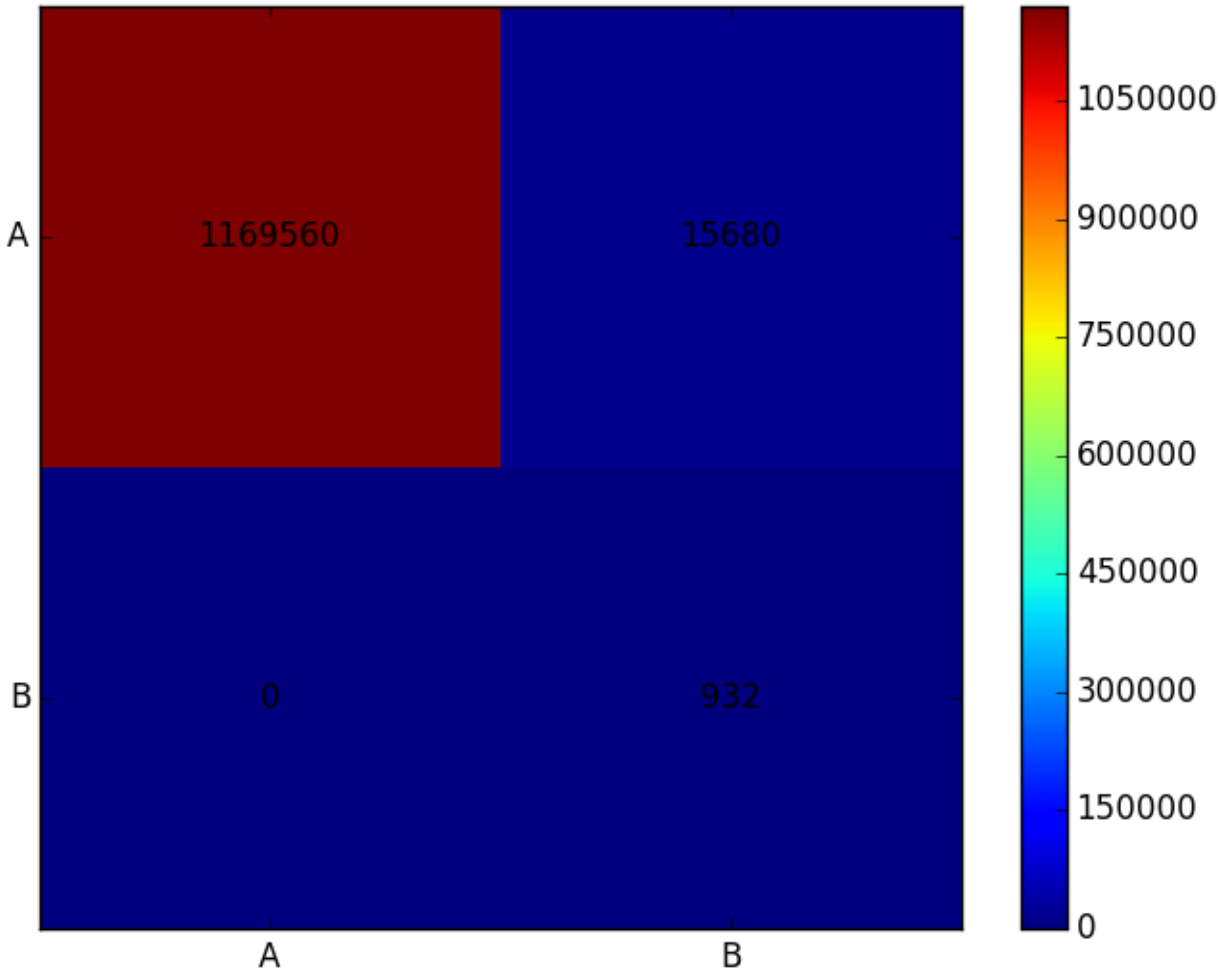
2.4 Oversampling

In an attempt to solve the class imbalance issues, I decided to try a method suggested in [3] to oversample from the underrepresented class. I set up a parameter c and created different datasets from my training set where the positive examples were sampled 2^c times. I then decided to try this method on the ensemble classifier since it gave me the better results of the 3 methods I tried out. I used 10-fold cross validation using the area under the curve metric to determine the best value for c and found that 5 seemed to be a good compromise. This brings the number of positive examples in the dataset from a couple of thousands to well over a million. As we can see from the confusion matrix obtained by predicting on the test set, oversampling allows us to significantly improve on the number of positive examples being picked up.In fact, we have no false negatives. This is especially

Table 4: choosing

	1	2	3	5	7
AVG AUC	0.75	0.77	0.82	0.85	0.83

interesting considering that the class imbalance is still present in the test set.On the other hand, the number of false positives is about 100 times larger than the number of true positives we get.



2.5 Discussion of the Results

Clearly, the biggest issue encountered when dealing with this type of data is the class imbalance. Overall, I would argue the results are better when we use oversampling than without but we are still far from the performance that can be found in the literature. We could argue I suppose that 15279 false positives is not bad when considering we have over 200000 examples in our test set.

However, in the case of real-time bidding, this means that for every ad placement we are interested in bidding on (the ones classified as positive) only 1 of them is actually going to result in a click. This can lead to a lot of money being wasted on auctions that we should not be participating in. On the other hand, when looking at the dataset that we have, we see that the average CTR of the bidders on the ad placement that they win are quite low, usually less than 1/100. For example, in the testing set, we have only 600 or so positive examples for 200000 auctions. This means that the majority of the ads that are bought by the bidders never convert into a click. Therefore, I would argue that a high false positive rate is actually quite acceptable when taking into consideration the context.

3 Building a Real-Time Bidder

In this section, I will cover my project for comp 762. My goal was to review the literature and test methods that can be used to build real-time bidders in the context of an ad marketplace. In the previous section, we decided that the KPI we were going to use to evaluate our learner is the number of clicks that our bidder obtains throughout an auction.

3.1 Expressing the auction as an MDP

First we need to explain how to translate the system defined by a second price auction into an MDP. In order to define the MDP, we first need to define what each state in the MDP corresponds to. An MDP in the context of a series of auctions is defined by a tuple(t,b,x) where t is the number of remaining auctions in the series, b is the remaining budget that the advertiser/agent has and x is the feature vector for the current ad placement that is being bid on. The start state for the bidder is always (T,B,x') where T is the number of auctions in total, B is the starting budget and x' is the first item to be bid on. We will be considering a second price auction system, meaning the bidder who wins the auction is the one with the highest bid, however the winner only needs to pay the price of the second price.
The reward from a taking action a at state(t,b,x') is going to be 1 if we observe that the ad was clicked on, 0 otherwise. Note that I am not including the payment amount in the reward function. I am working under the assumption that because we have a limited budget, it will be in the bidders best interest to learn to use small bids rather than large ones when possible in order to save up money for future auctions. Depending on the results of the auction, we will transition to state (t-1,b-c,x) (if we won the auction and the second price was c) or (t-1,b,x) (if we didn't win the auction).

3.2 Q-Learning

The first method that I wanted to evaluate on this problem is the Q-Learning off-policy method we learned of in class. The advantage of Q-Learning over the Dynamic Programming Method in

[1] is that it allows us to learn which actions to take given a state in an online fashion. Therefore, we do not need to estimate the bidding landscape (the probability of winning an auction given a certain bid).

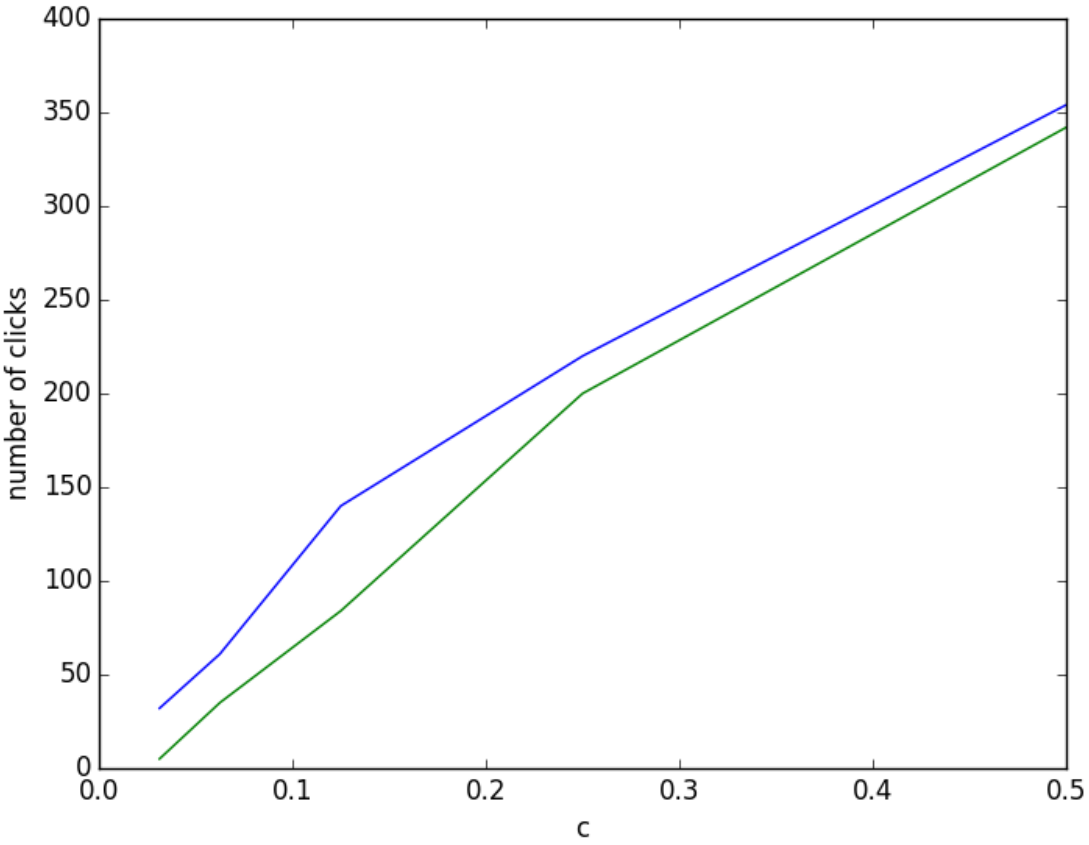
In the previous section, we discussed different methods that can be used to estimate the probability that a user will click on our ad should we win the auction. In this section, we will assume that our bidders have access to a function that allows them to estimate their CTR.

In order to test the Q-Learning method, I designed a synthetic auction. The auction is made up of 3 bidders. There are 3 possible types of ad placement. Each bidder is assigned a function that takes as input the type of the ad placement and returns the CTR for that bidder. The functions are different for each bidder to reflect different preferences for the type of the ad placement.

The first hypothesis that I wanted to test was that in a setting with a limited budget, a bidder that utilizes Q-Learning is going to perform better than a bidder that uses the common LIN method. The action taken by a LIN bidder given an ad placement x is:

$$aLin(x) = b_0 * \frac{CTR(x)}{AvgCTR} \tag{3}$$

In order to test this hypothesis, I set up two auctions, one where all three of the bidders used the LIN method and one where one of the bidders used Q-Learning to learn which actions to perform and the other two used LIN. In order to train the Q-Learning bidder, I performed 10000 episodes of 100 auctions. The budget was the same for all three bidders. It was 3*c*T, where c is varied to limit the budget of the bidders. I looked at two different situations, one where one of the LIN bidders was in direct competition with the player I was trying to optimize and one where there is less competition. In order to do this, I simply selected CTR functions that fit the requirements (high CTR for the same type of ads for competition and high CTR for different type of ads for no competition). I gave my bidder a slightly higher CTR than the opponent in order to illustrate one situation where I believe learning from the environment is more beneficial than having a linear decision function for the action. In my slides for the presentation, I talked about the case for no competition and argued that the Q-Learning bidder performed better than the LIN bidder. However, after retraining the bidder multiple times, I found that it wasn't always the case and that the improvement was not very significant. The competitive case however produces more interesting results. The following graph is the cumulated reward obtained by both the Lin Bidder and the Q-Learning bidder in a test run of 50*100 auctions in the case where my bidder is in competition with some other bidder for ads of type 1.



Graphic 1: Comparaison of Q-Learing(blue) vs Lin(green)

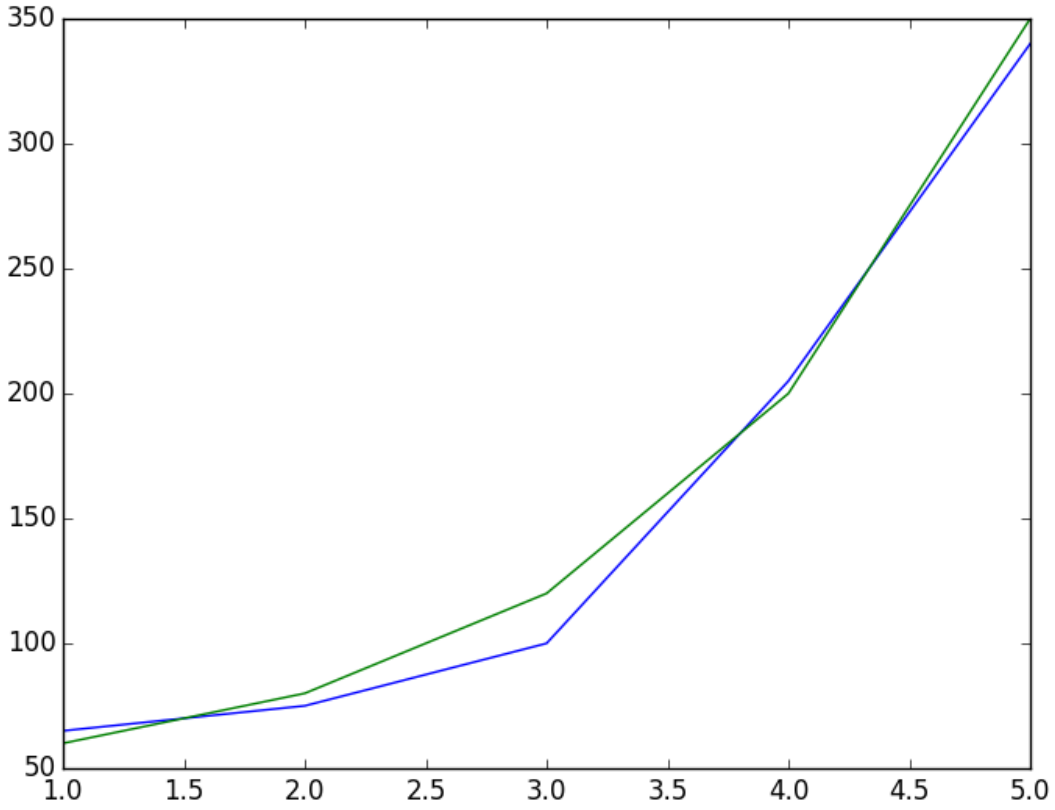
As we can see from the graph, in a competitive environment, the Q-learning bidder performs better than the LIN bidder in the case where the budget is restricted. This is most likely because, when the budget is restricted, the LIN bidder is bidding large amounts on ad placements where the second price is also very high due to the competition. Therefore, the budget is quickly exhausted and the bidder performs poorly. A bidder that learns directly from the environment however is able of identifying these sorts of traps and manage it's budget better, for example by waiting until the end of the auction to bid. With a larger budget however, I see no improvements between LIN and Q-Learning.

3.3 Q-Learning with modified state space

One obvious flaw in the tabular Q-Learning case is that in the case where the ad placement is described with many features, the state space will be very large and it will be likely that in practice, the algorithm will encounter a state that has never been visited (and therefore no action was ever learned for that particular situation). In the case where T or B are very large, [2] suggest we have the option of simply dividing up the auction into many smaller auctions. They also suggests

defining a set of neighbors for every state and if we encounter a state that has never been visited, we can use the action of one of the neighbors. For my second hypothesis, I wanted to test another option for reducing the state space which was to replace x in the state space by the single value CTR(x).

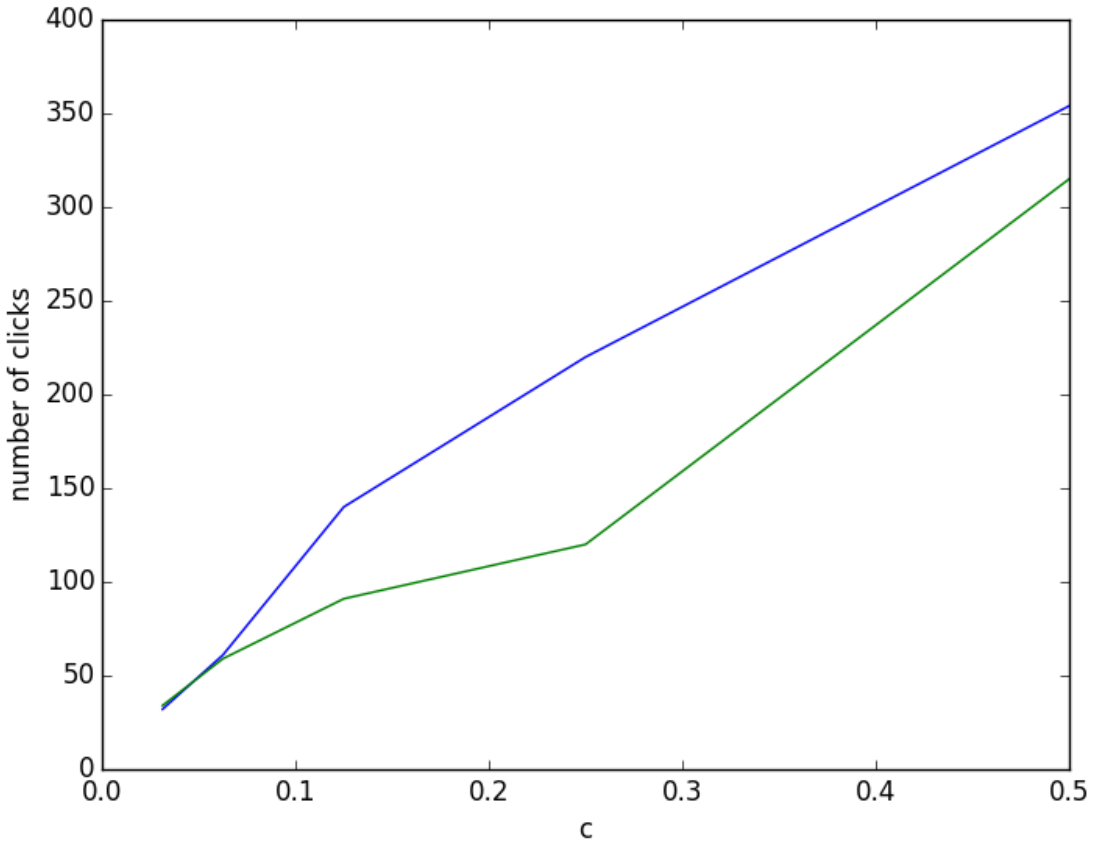
We can observe that whereas the LIN bidders make use of the CTR in order to decide how much to bid, the Q-Learning bidder so far does not. By replacing the set of features x by the CTR of x, we can greatly reduce the state space. In order to test whether this idea is good, I compared two bidders, one that uses x in its state-space and one that uses the CTR of x (I actually grouped the CTR into 4 groups, from 0.0 to 0.05, from 0.05 to 0.1, from 0.1 to 0.15 and from 0.15 to 1) in the competitive case described above. As we can see from the following graph, both bidders perform similarly. Therefore it seems appropriate to use the CTR to describe the state space.



Graphic 2: Comparaison of Q-Learing with features(blue) vs Q-Learning with CTR(green)

However, in my example, the bidders have the advantage of knowing the exact CTR of x. This is not a realistic expectation to have in the real world. Therefore, we also need to consider a case where we introduce some error in the CTR function. From the results that I obtained in the first

section of the report, we saw that CTR predictions tend to be more optimistic than usual if we want to be able to detect all the positive examples. Therefore, I also studied the case where the CTR of x predicted by the bidder is modified with noise. This was done by using a Gaussian function with mean 0 and variance of 0.05. As we can see from the following graph, this does cause some problems for the bidder. The performance is worse than before we added the noise when the budget is large. Perhaps this is due to the bidder having to be more conservative with its bids (only bid when the CTR is high in order to make sure that it is actually high). In the case where we have a lower budget, no change is observed which makes sense since the bidder would already need to be conservative with the budget in order to maximize revenue.



Graphic 3: Comparaison of Q-Learing with CTR no noise(blue) vs Q-Learning with CTR with noise(green)

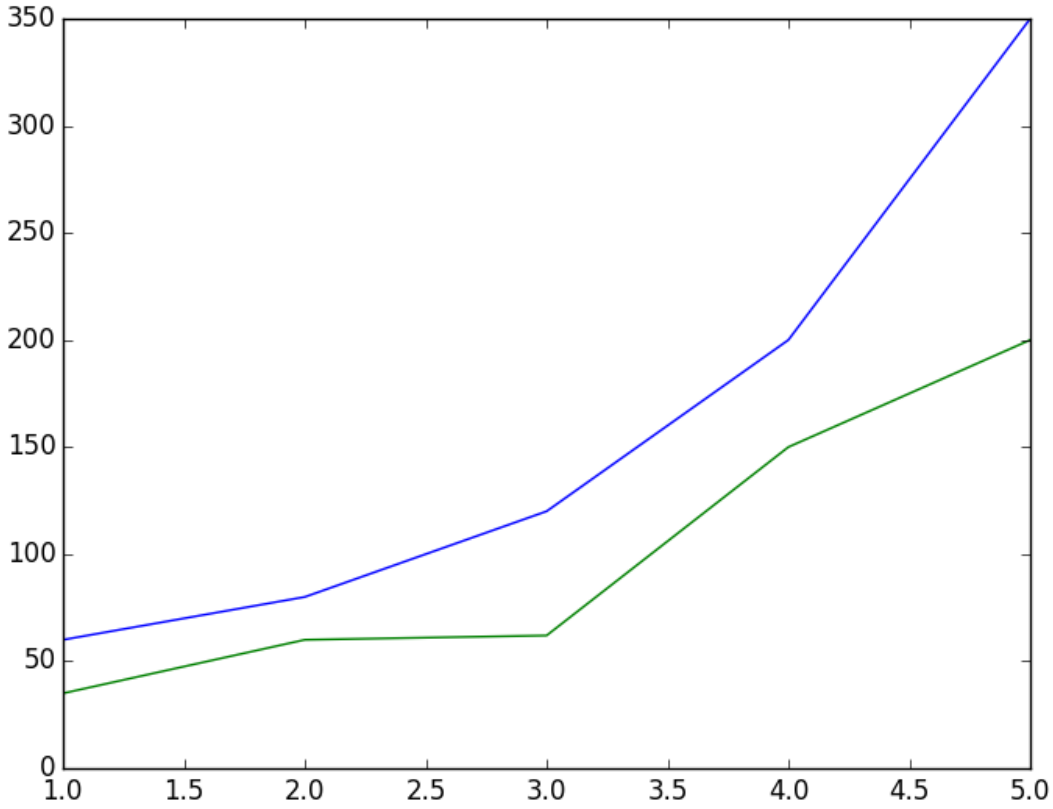
3.4 Generalizing to a Multi-Agent system

Throughout the two previous sections, we used two LIN bidders as the our opposing bidders. These Lin bidders have a fixed policy. Although in practice it appears to be sufficient to assume that the opposing bidders have a fixed policy, I was also interested in looking at a case where every bidder

has is using a dynamic policy, capable of adapting to the environment.
Specifically , I decided to look at what would happen if every bidder was using the Friend-Q algorithm. The logic behind this choice is that poor performance in auctions can only be driven by competition. When there is no competition, it is easy to get a lot of reward for cheap. However, when bidders compete, the second price will be large and this will force the bidders to pay a lot of money for the same rewards, limiting their ability to bid in other auctions.
We know fortunately that second price auctions are susceptible to collusion, meaning bidders can agree to not compete and distribute the different prizes among themselves. My hypothesis is that this approach constitutes a coordination equilibrium, a Nash equilibrium that finds the optimal reward for all the players. In [4], it was shown that by using Friend-Q, bidders who do not communicate will converge to a coordination equilibrium. As a reminder, the update rule for Friend-Q is with three bidders:

$$Q(s, a1, a2, a3) = reward + \alpha * (max_{a1', a2', a3'} Q(s', a1', a2', a3') - Q(s, a1, a2, a3)) \tag{4}$$

In practice, the results were not as good as I expected. I compared the cumulative reward obtained by one of the Friend-Q bidders to the reward obtained by a Q-Learning bidder playing against opponents with a fixed policy and the results were clearly worse. This means that either I was wrong about the presence of a coordination equilibrium in the game, or the training was too short to allows for the algorithms to converge to a good solution.



Graphic 4: Comparaison of Q-Learing/Lin(blue) vs Friend-Q(green)

4 Conclusion

In conclusion, these two projects were centered around using machine learning and reinforcement learning to build real time bidder for the ad marketplace. I first showed that the difficulty in predicting the CTR of an ad lies in fact that the data used suffers from class imbalance. I studied various methods to try to address the problem and found that the best solution was to use over-sampling combined with a boosted decision tree learner. Despite this, I still have a large number of false positives in my predictions.

For the reinforcement learning project, I looked into using Reinforcement Learning Methods to build a bidder that is able to learn to perform better than the usual Lin classifier. I decided to use Q-Learning since it requires no policy and can be generalized to the multi-agent case with Nash-Q. I showed that in a competitive environment, using Q-Learning can allow the bidder to learn to avoid competition in order to save money against competitors. I also showed that we did not need to use the feature vector of the ad in order to represent the state but could get away with using the CTR as long as there is not too much error between the true CTR and the predicted CTR

of the bidder. Finally I attempted to show that using Friend-Q could provide better results by teaching the bidders to collude, however I was unsuccessful in doing so. In terms of feature work, I would like to explore more different methods proposed in [3] to reduce the dimensionality of the data which can be problematic in real-life. This can be done by taking advantage of online version of the algorithms presented in part 1. Furthermore, I would like to try other methods presented in class to teach the bidder to bid correctly. Especially, I would try to use Double-Q learning to perhaps speed up the training process in the case it is affected by maximization bias.

5 packages and datasets

In order to make the predictions from the first part of the report, I used the scikit-learn package. Furthermore, the iPinYou dataset can be found here Finally, I used the code from here to merge the data files from iPinYou and extract the data for advertiser 300. My code can be found on the github webpage here.

References

[1] Weinan Zhang *Real-Time Bidding Benchmarking with iPinYou Dataset.*

[2] Han Cai *Real-Time Bidding by Reinforcement Learning in Display Advertising.*

[3] H. Brendan McMahan *Ad Click Prediction: a View from the Trenches.*

[4] M.Littman *Friend or Foe Q-Learning in General Sum Games.*