CS674 Project
Understanding Net Advertising

Raunaq Malhotra............. Y5370
Utkarsh Upadhyay Y5488
Subhonmesh Bose Y5458

April 19, 2008

Contents

1 Understanding the problem statement

As we understand the problem statement, we have to solve 2 problems here. The actual problem statement as given by *Komli.com* requires us to find *features* that result in higher *click_through_rates* which is essentially a regression problem. The project however requires us to classify individual records as cliked and not-clicked. Thus there are 2 problems to solve:

- Regression
- Classification

Here we attempt to focus on the Classification, but as a part of the classifier itself we incorporate a simple regressor model. Thus we do not differentiate between these 2 facets of the problem from here on and give the structure of the classifier basically.

2 Motivation to the design

Each record contains a set of some id's and a set of features. The id's are enumerable attributes and the features are numerical (double) data.

- ***List of id's:*** *pub_id*, *site_id*, *adserver_id*, *adserver_optimizer_id*, *user_guid*, *page*, *user_ip_addr*.
- ***List of features:*** *ad_height*, *ad_width*, *bg_color*, *border_color*, *link_color*, *url_color*, *text_color*.

Now we have just 223 different such features present in the data of 5391436 records. We also note that these 223 records are not complete in the sense that there are feature vectors for which some colors are missing. In around 85 different feature vectors among the 223, all color columns show *NCOLOR*. We first perform a simple conversion of the color bit-string to the RGB values and fill in the *NCOLOR*'s with the average R,G and B values for the particular parameter obtained over records which are complete w.r.t that particular parameters. Thus we obtain 15 ($= 5 \times 3$) parameters. Please note that we do not keep *ad_id* as an id in the list because it has a one-one correspondence to the distinct features present in the data.

We now come to the design of the classifier. With so few distinct features, it is obvious that the classification of records cannot only base on the feature-vector. The id's do have a say on the particular classification of a record. To model this idea, we try to build up the following 2 units:

1. Probability of getting clicked, given the id's
2. Probability of getting clicked, given the feature vectors

Finally we try to build a classifier based on these 2 probabilities. Now we analyze the id's before we go on further with the design of the classifier.

The following table enlists the ids and also enlists how many different such id's have been found in the text.

id	no. of distinct id's
pub_id	84
site_id	104
$adserver_id$	5
adserver_optimizer_id	5
user_ip_addr	6
user_guid	869478
page	167489

Now to calculate probability of clicked, given id's, we first find the mutual information to giude us through the calculation procedure. The table is presented below.

attribute	I(clicked; attribute)
user_ip_addr	0.0000001596
adserver_optimizer_id	0.0000257051
adserver_id	0.0009708456
pub_id	0.0034722215
site_id	0.0035926729

With *user_ip_addr* we find that the mutual information is almost negligible. Thus we simply neglect the ip-address. Moreover we note that there are 869478 *user_guid*s and 167489 *page*s. The sheer numbers deter us from choosing them as a valid basis of splitting our decision tree. Interestingly there were just 4 users with more than 5000 records in the dataset, which definitely gives us an idea that the dataset we have does not contain many users whose individual trends would matter anyways since training over just 5000 data for a particular user might not actually give us good results due to the small training set of just a 5000. If we lower the threshold of 5000 for views of a user, we find that there are 693 different users who have featured in the table more than 300 times. Thus modelling a user would require us too many different classifiers at leaf nodes of a decision tree. So we rather leave this parameter out as well for our classification. Similar is the case with the *page* paramater which we also decide to leave out.

2.1 Calculation of the the probability of clicked, given id's

Here we simply take a counting approach. We consider $(pub\_id, site\_id)$ as a tuple and $(ad\_server\_id, ad\_server_optimizer_id)$ as another tuple. We could have taken all the 4 into one 4-tuple and worked, but then there are only 803 such distinct 4-tuples which leads us to give non-zero predictions only on the tuples we have encountered in the training set. That kind of reduces any kind of generalization. Now we maintain simple matrices (84×104) sized one for $(pub\_id, site\_id)$ and (5×5) sized one for $(ad\_server\_id, ad\_server_optimizer_id)$. Here we go through the learning set and simply keep a count on how many times various distinct tuples have been viewed and how many times each such distinct tuple has been clicked. Thus probability of each such tuple being clicked becomes

$$p(\text{Clicked}|\text{a tuple of id's}) = \frac{p(\text{Clicked, tuple of id's})}{p(\text{tuple of id's})}$$

Thus we find 2 such probabilities corresponding to 2 such tuples.

2.2 Predicts a *click_through_rate* for a feature vector

Here we tried out various approaches. We briefly touch upon all the approaches taken before we describe the process we used finally:

ANN This gives us fairly random results. Moreover as the *click_through_rate*s are anyways so small, the never-clicked ones get a predicted value pretty near the predicted values for the clicked ones.

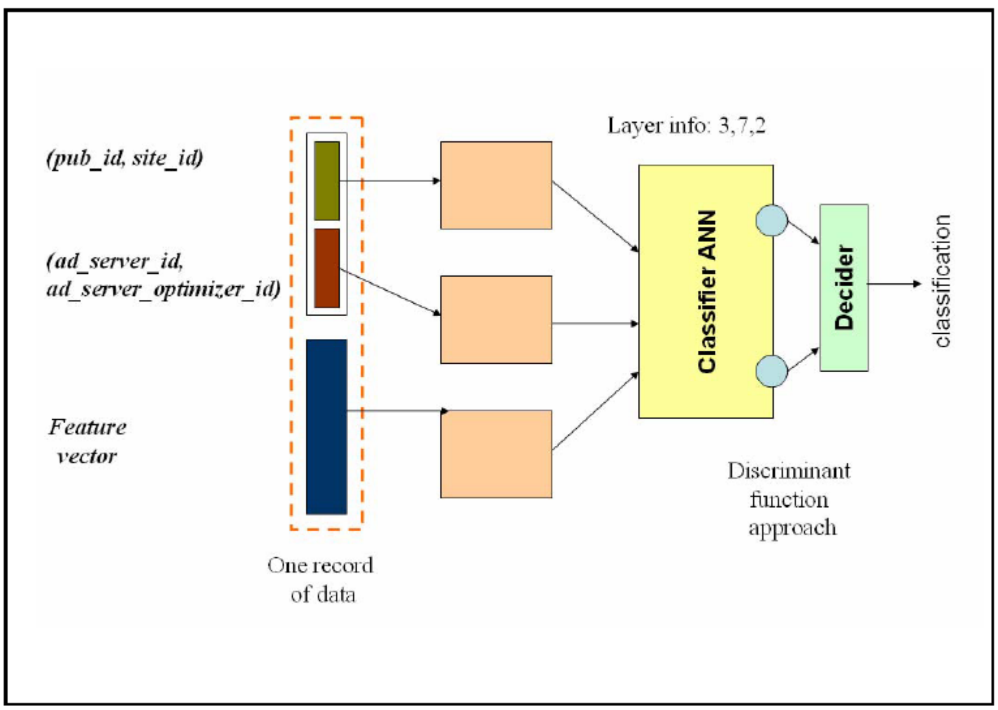
RBF-ANN The radial basis function ANN in Matlab is well-known for smooth curve-fitting. But this approach suffered similar results as the simple ANN.

SVM-ANN Here we tried to seive out the never-clicked feature vectors from the at-least-clicked-once feature vectors through an SVM and then tried to train the ANN over the *click_through_rate*s of the at-least-clicked-once features. The results were even worse as we had 2-fold errors here. Misclassification due to SVM followed by a not-so-good ANN regressor gave us similar bad results here as well.

SVM regressor The SVM library that we used in Matlab (simpleSVM) had a feature of SVM regressor which we tried to use. This too did not yield good results.

k-nearest-neighbor Finally we took this approach. Firstly we changed the feature space for distance calculation. We took all the ${}^5C_2$ combination of contrasts and 5 brightnesses of the 5 colours that we have and then normalized all the inputs to $[0, 1]$. Then we defined simple Euclidean distance between 2 feature vectors. The contrast and brightness calculations are done in standard ways. Now when we encounter a feature-vector, we simply choose the k-nearest neighbors and take a weighted mean of their *click_through_rate*s, weighted over their number of views in the training set. This gives decent results with $k = 1, 2, 3$. For demo purposes we simply take $k = 1$ for faster running, otherwise we can go for $k = 2, 3$ without much problem. Thus we have a regressor for the classifier.

2.3 The structure of one classifier



In the ANN, thus we get 3 inputs. Now we run a simple ANN model with 3,7,2 layers. There are 2 outputs basically which behaves akin to discriminant functions and we take a decision in favour of clicked if the second output is greater than the first output and not-clicked for the other way round. This ANN required us to suitably scale the inputs, which almost formed the backbone of training the ANN. The target was $(1, -1)$ for not-clicked cases and $(-1, 1)$ for clicked cases and we thus tuned our ANN.

3 The incremental nature of our learning

Firstly we note that in around 53 lakh data, we have only 223 distinct feature vectors. Thus we hope that the distinct number of feature vectors we obtain in the data is pretty low and we can thus maintain a set of distinct feature vectors encountered. Now we comment upon the incremental nature of our training. The first unit that calculates the probability given the id's is obviously incremental as simply require counting. When we encounter new data in the training set, we simply increment the suitable array element for the matrices. The ANN is iteself an incremental structure and the regressor also works only on the set of distinct feature vectors which thus gives us a fully incremental nature of training.

4 Bag of Classifiers

Due to memory and time constraints, we do not obviously train one classifier over the entire data. Rather we have built 7 classifiers that have been trained on parts of the dataset and take a majority vote on their classification for the final classifier. Now this gives us a couple of advantages:

1. Generalization: Since the dataset has not been fully used for the training of a classifier, each classifier infact generalizes over the entire dataset and thus this method gives us an edge over any algorithm that uses 1 classifier trained over the entire dataset in terms of generalization performance.

2. Error reduction: Theoretically the bag-of-classifiers approach reduces the error.

3. Parallel training possible: We do not require to train on such a large dataset and let individual classifiers parallely learn over different part of the dataset that leads to faster build-up of the classifiers.

5 Biasing the dataset for training

There are only 41443 records out of 53 lakh odd data, which have been clicked. Thus a random sampling would end us with data having $\approx 1\%$ clicked data. Any classifier would thus give us a blanket not-clicked as that leads to only 1% error-rate. Thus during training we bias the dataset by intentionally using 20%,30% and 40% clicked data in the training sets. This gives us considerably better results.

6 Testing the classifier

After training, we took 4 sample to test the results on. In one we had 20% bias over the dataset, 2 were randomly sampled from the entire dataset and 1 had only clicked ones. Our classifier performed reasonably well on the 20% dataset.

	20% bias	Random Data 1	Random data 2	100% clicked
True Positives	366	13	13	*941*
False Positives	287	389	409	*0*
True Negatives	3700	4944	5072	*0*
False Negatives	646	29	24	*4058*

The data sets were of the approximate size 5000 entries. Using the entries it can be seen that our classifiers work better when they are tried over data that has almost the same bias over which our classifiers have been trained. Hence, the extreme error rate over data that is heavily clicked.

7 Bad performance on the set where we have same *user_guid*

Also, we have noted (from the data given to us to test our program on) that the uncertainity in the *pub_id, site_id* tuple is almost zero given the *user_guid*. So was the case in all the three users given to us for testing. Our regressor outputted nearly a positive on all the cases, which clearly shows that the weight of the *(pub_id, site_id)* tuple is very high in the neural network.

Here, we wish to add that we were tempted to replace all the '1's with '0's and vice versa for these test case. That would have given us a much lower error rate. We would like to bring this to your notice that the results could have been easily manipulated because of the freedom given to us. Moreover the test set being a subset of the original, a lookup could have been very easily performed giving any arbitraty accuracy. Hence, if possible the code should be examined and the final results should be taken with a pinch of salt.

8 Conclusion

Overall, the project was an enthralling learning experience. We all learned plenty from the project, ranging from handling a MySQL database to how to tune the parameters for a SVM to how to make MATLAB codes perform *slightly* faster. It seems that the project went far beyond just machine learning. Though we finally ended up using a very simple regressor (k-Nearest neighbour), it was not before we were through with trying to tune the ANNs and the SVMs to the maximum. Hence, the most imoprtant lesson that we learnt from the project was the one may need to do more than just *throw* any machine learning technique on a problem.

9 Acknowledgements

Foremostly we would like to thank Prof. Harish Karnick for giving us the insights into Machine learning and Manu Bansal for helping us with the nitty gritties of the project. Also, we took references from various sources (books, website, papers, tutorials, etc.) which we find a little diffuclt to cite here.