STAT 157
Fall 2014

Eileen Li, Kevin Shieh
Interian

Final Report

Introduction

As the world is becoming increasingly automated, so has advertising. In working with search engine queries, companies are able to better show users their ads based on what the user searched or has searched in the past. This is extremely powerful since by simply searching on a search engine, the user already expresses interest the query material and is in theory will find ads relevant to said material more relevant to themselves. In a more 2014 and specific example, new technology companies are starting up all over the world and in Silicon Valley in particular. These companies often find themselves challenged to monetize their product. In the face of such problems, many companies turn to targeted ads.

The problem we are presented with is to find a way to most accurately predict the click-through rate (CTR) on advertisements. Given a large number of query instances, the goal is to train machine learning models such that we have the highest prediction rate on a set of test instances. An instance is described as an individual session in which a user made a search on Tencent's proprietary search engine, soso.com, and was presented with an ad under specific circumstances. The circumstances are primarily the depth of the ad, the number of ads shown or impressed in a session, and the position, or the order of the ad in the list of ads shown. Each instance also comes with the following features: click, impression, displayURL, adID, advertiserID, depth, position, queryID, keywordID, titleID, descriptionID, userID. The last five features have corresponding files that provide more details. For example, in the corresponding userID file, we are also provided with the user's gender and age. In the queryID, keywordID, titleID, and descriptionID files, there is a mapping from each respective ID to a list of tokens. These tokens are representations of words. I separate the total data such that 20% is the test set, 20% is the validation set, and 60% is the training set. The problem and data is provided by the 2012 KDD Cup competition.

The rest of the report will be dedicated to first explaining the features we chose to use in our models, the rationale behind the models we used, methods of data preparation and training and testing our models, results, responsibilities of members, and finally a post-mortem of the entire project.

Features

Feature selection is the backbone of a machine learning model, as this is what the models use to predict outcomes. For the issue of CTR, we thought about the problem from a user perspective and asked ourselves what would most differentiate whether or not a user clicked on an ad in a particular session? We decided on two main factors: relevance of the ad to the query and user information. The former is straightforward, since the query obviously is relevant to the user and therefore, the content of a related ad should be more relevant. The latter should also be a clear choice. Especially today where the gap in technology usage between younger and older generations grows, we figured the behavior towards ads would differ as well. With the

advent of Adblock and the feelings of entitlement to free and ad-free online experiences, we predicted that younger users would be less likely to click on ads.

For user information, we used the following groups to categorize each instance into one of the following twelve groups by finding the cross product between the two genders and the age groups.

Males

- (0, 12]
- (12, 18]
- (18, 24]
- (24, 30]
- (30, 40]
- greater than 40

Females

- (0, 12]
- (12, 18]
- (18, 24]
- (24, 30]
- (30, 40]
- greater than 40

For the relevance of an ad to a query, we utilized the additional files. We had three different metrics for ad to query relevance.

$\frac{\#\ query\ tokens\ in\ title\ tokens}{\#\ query\ tokens}$ where title tokens are words in the ad title.

$\frac{\#\ query\ tokens\ in\ description\ tokens}{\#\ query\ tokens}$ where description tokens are words in the ad description.

$\frac{\#\ query\ tokens\ in\ purchased\ tokens}{\#\ query\ tokens}$ where purchased tokens are words purchased by the advertiser.

Models

We considered a number of different models to solve this classification problem. The first we looked at was the simple Decision Tree Classifier. How this model works is that given features $F_1, F_2, .., F_n$, it will build a decision tree where at each step, it goes down the path for given feature that has the highest probability of occurring. This is a very basic model that is easy to set up and requires minimal data preparation. We built this decision tree using the age and gender feature. However, we found that since there are much more ”no click”s than ”click”s, this model would fail and end up predicting ”no click” on every test instance.

The next model we considered was the Forests of Randomed Trees, which is an extension of the Decision Tree Classifier. This model works by randomly creating many decision trees, each of which has a random subset of the features that we are predicting on. It then averages the results of all generated trees and outputs that as the answer. However, this model did not work well for our features since we didn't have enough features to actually make this technique viable. With only a feature features, it becomes roughly equivalent to the basic Decision Tree Classifier.

Finally, we settled on a Bernoulli Naive Bayes model. This model worked well for us for various technical reasons that will be detailed below in the "Implementation" section. Naive Bayes can be summed up by a straightforward equation:

$$classify(f_1, ..., f_n) = \underset{c}{argmax}\ p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).$$

This basically says that for a set of features, $F_1, ...F_n$, the best prediction is the value c that maximizes the right hand side of this equation. In our situation with title token similarily, C takes on value *click* or *noclick*.

$$p(Click)p(Title\ token\ similarity = appropriate\ bucket | C = c)$$
$$p(Click) = \frac{total\ \#\ clicks}{total\ \#\ impressions}$$
$$p(Title\ token\ similarily\ =\ appropriate\ bucket | C = c) = \frac{\#\ clicks\ in\ bucket}{total\ \#\ clicks}$$

$$p(No\ click)p(Title\ token\ similarily = appropriate\ bucket | C = c)$$
$$p(No\ click) = \frac{total\ \#\ impressions - total\ \#\ clicks}{total\ \#\ impressions}$$
$$p(Title\ token\ similarity\ =\ appropriate\ bucket | C = c) = \frac{\#\ impressions\ in\ bucket - \#\ clicks\ in\ bucket}{total\ \#\ impressions\ - total\ \#\ clicks}$$

$$if\ p(Click) \geq p(Noclick), predict\ click$$
$$else\ predict\ no\ click$$

We applied this Naive Bayes model to the title, description, and purchased similarity features.

To measure the performance of our model, we used the area under the curve (AUC) of a receiver operator characteristic (ROC) graph. The AUC can be interpreted as the probability of ranking a randomly chosen click instance over a randomly chosen no click instance. If the AUC is 0.5, that means the classifier is just guessing completely. If the AUC is 0.0, that means that the classifier is predicting the opposite value every time. And if the AUC is 1.0, that means the classifier is making the correct prediction every time. We use this to measure how well our classifier is doing.

Implementation

We started by separating the main data file into much small and more manageable chunks so that we could test things locally and quickly. Upon doing this, we first tackled the user demographic features. We used the `sklearn` package in Python to build Decision Trees. However, we were unable to train the model to predict anything outside of a "no click". This resulted in an AUC of 0.5 at all times. We figured that the time we invested into learning how `sklearn` was too much already and that our time would be best spent working on the token features.

We used Hadoop's Elastic Map Reduce (EMR) to aggregate and transform our data. Below is the pipeline to create the appropriate training data for our title and query token similarity feature. The pipeline is very similar for description and purchased similarity features. The output files can be located by prepending `s3://stat157-uq85def/home/kevinshieh` to the directory names. `*.txt` files can be found in `s3://stat157-uq85def/shared/track2/`. A comprehensive list of data and code is located at

the end of this report in the ”Index” section.

1. Join instances and query tokens on `queryID`
   Mapper: `join_query_mapper.py`
   Reducer: `join_query_reducer.py`
   Input: `training.txt, queryid_tokensid.txt`
   Output: `join_query_out/`
   Output format: `clicks impressions ... query_tokens`
   EMR:

```
-files s3://stat157-uq85def/home/kevinshieh/code/join_query_mapper.py,
s3://stat157-uq85def/home/kevinshieh/code/join_query_reducer.py
-mapper join_query_mapper.py -reducer join_query_reducer.py
-input s3://stat157-uq85def/shared/track2/training.txt,
s3://stat157-uq85def/shared/track2/queryid_tokensid.txt
-output s3://stat157-uq85def/home/kevinshieh/join_query_out
```

2. Join query-joined instances and title tokens on `titleID`
   Mapper: `join_title_mapper.py`
   Reducer: `join_title_reducer.py`
   Input: `titleid_tokensid.txt, join_query_out/`
   Output: `join_query_title_out/`
   Output format: `clicks impressions ... query_tokens title_tokens`
   EMR:

```
-files s3://stat157-uq85def/home/kevinshieh/code/join_feature_mapper.py,
s3://stat157-uq85def/home/kevinshieh/code/join_feature_reducer.py
-mapper join_feature_mapper.py -reducer join_feature_reducer.py
-input s3://stat157-uq85def/shared/track2/titleid_tokensid.txt,
s3://stat157-uq85def/home/kevinshieh/join_query_out/part*
-output s3://stat157-uq85def/home/kevinshieh/join_query_title_out/
```

3. Calculate similarity ratio
   Mapper: `similarity_mapper.py`
   Reducer: `similarity_reducer.py`
   Input: `join_query_title_out/`
   Output: `similarity_title_out/`
   Output format: `clicks impressions similarity_ratio`
   EMR:

```
-files s3://stat157-uq85def/home/kevinshieh/code/similarity_mapper.py,
s3://stat157-uq85def/home/kevinshieh/code/similarity_reducer.py
-mapper similarity_mapper.py -reducer similarity_reducer.py
-input s3://stat157-uq85def/home/kevinshieh/join_query_title_out/part*
-output s3://stat157-uq85def/home/kevinshieh/similarity_title_out/
```

4. Label instances to split into train, validation, and test sets
   Mapper: `label_data_mapper.py`

Reducer:label_data_reducer.py
Input: similarity_title_out/
Output: similarity_title_labeled_out/
Output format: train clicks impressions similarity_ratio
EMR:

```
-files s3://stat157-uq85def/home/kevinshieh/code/label_data_mapper.py,
s3://stat157-uq85def/home/kevinshieh/code/label_data_reducer.py
-mapper label_data_mapper.py -reducer label_data_reducer.py
-input s3://stat157-uq85def/home/kevinshieh/similarity_title_out/part*
-output s3://stat157-uq85def/home/kevinshieh/similarity_title_labeled_out/
```

5. Separate instances into train, validation, and test directories
   Mapper: separate_train_mapper.py
   Reducer:label_data_reducer.py
   Input: similarity_title_labeled_out/
   Output: title_train
   Output format: train clicks impressions similarity_ratio
   EMR:

```
-files s3://stat157-uq85def/home/kevinshieh/code/separate_train_mapper.py,
s3://stat157-uq85def/home/kevinshieh/code/label_data_reducer.py
-mapper separate_train_mapper.py -reducer label_data_reducer.py
-input s3://stat157-uq85def/home/kevinshieh/similarity_title_labeled_out/par
-output s3://stat157-uq85def/home/kevinshieh/title_train/
```

6. Group instances on similarity ratio into 0.1 width buckets
   Mapper: group_by_rate_mapper.py
   Reducer:group_by_rate_reducer.py
   Input: title_train
   Output: groupby_title
   Output format: similarity_ratio clicks impressions
   EMR:

```
-files s3://stat157-uq85def/home/kevinshieh/code/group_by_rate_mapper.py,
s3://stat157-uq85def/home/kevinshieh/code/group_by_rate_reducer.py
-mapper group_by_rate_mapper.py -reducer group_by_rate_reducer.py
-input s3://stat157-uq85def/home/kevinshieh/title_train/part*
-output s3://stat157-uq85def/home/kevinshieh/groupby_title/
```

7. simple_combine.py
   Input: groupby_title
   Output: final_title.txt
   Because the output in groupby_title was split into multiple files, I needed to simply merge them into a single file.

8. simple_model.py
   Input: title_train.txt, title_test.txt

This is where I train the Naive Bayes model using `title_train.txt` and test the model classification by feeding in some validation or test data. Here, I modeled my Bernoulli Naive Bayes by using the equations mentioned above in the "Models" section. I test the test set here and calculate the AUC here.

Results

Our main three models were the Naive Bayes run on the similarity rates for title, description, and purchased keywords. Their results are outlined in the table below.

	Title	Description	Purchased
AUC	0.57568	0.50143	0.50052
% Correctly Predicted	0.69476	0.93258	0.95024

It turns out that the title similarity resulted in the highest AUC. However, the % of corrected predicted instances was lower than the description and purchased models. We believe the cause of this is that the description and purchased models ended up predicting "no click" for a vast majority of instances, resulting in a high % of correct predictions since there are much more "no clicks" than "clicks". For example, in the purchased model, there were 28434080 true negatives out of 29933503 total instances. Therefore, the true negatives alone accounted for 94.990% of the instances already. This seems slightly abnormal, but our reasoning is that users don't read descriptions of ads, but rather only focus for a brief few seconds on the title to decide whether or not they want to click on it.

These results may help us make more informed decisions about what ads will be clicked. It seems that when ad titles contain a large proportion of the keywords in a query, users are more likely to click on the ad.

Post-mortem

We could have improved on this project had we had more time and more people. Our team of 2 was unable to do as much work as desired on this project. With more time, we would have first tried making features from combinations of title, description, and purchased keywords. In addition, we would create models with the similarity ratio features as well as user features combined together. We would also explore more different types of models.

Index

s3://stat157-uq85def/shared/track2/training.txt
s3://stat157-uq85def/shared/track2/queryid_tokensid.txt
s3://stat157-uq85def/shared/track2/titleid_tokensid.txt
s3://stat157-uq85def/shared/track2/descriptionid_tokensid.txt
s3://stat157-uq85def/shared/track2/purchasedid_tokensid.txt
s3://stat157-uq85def/shared/track2/userid_profile.txt
s3://stat157-uq85def/home/kevinshieh/join_query_out/
s3://stat157-uq85def/home/kevinshieh/join_query_title_out/
s3://stat157-uq85def/home/kevinshieh/join_query_description_out/
s3://stat157-uq85def/home/kevinshieh/join_query_purchased_out/
s3://stat157-uq85def/home/kevinshieh/similarity_title_out/
s3://stat157-uq85def/home/kevinshieh/similarity_description_out/
s3://stat157-uq85def/home/kevinshieh/similarity_purchased_out/
s3://stat157-uq85def/home/kevinshieh/similarity_title_labeled_out/
s3://stat157-uq85def/home/kevinshieh/similarity_description_labeled_out/
s3://stat157-uq85def/home/kevinshieh/similarity_purchased_labeled_out/
s3://stat157-uq85def/home/kevinshieh/title_train/
s3://stat157-uq85def/home/kevinshieh/title_test/
s3://stat157-uq85def/home/kevinshieh/title_validation/
s3://stat157-uq85def/home/kevinshieh/description_train/
s3://stat157-uq85def/home/kevinshieh/description_test/
s3://stat157-uq85def/home/kevinshieh/description_validation/
s3://stat157-uq85def/home/kevinshieh/purchased_train/
s3://stat157-uq85def/home/kevinshieh/purchased_test/
s3://stat157-uq85def/home/kevinshieh/purchased_validation/

https://github.com/ucb-stat-157/Team-2/tree/master/code/join_query_mapper.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/join_query_reducer.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/join_feature_mapper.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/join_feature_reducer.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/similarity_mapper.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/similarity_reducer.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/label_data_mapper.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/label_data_reducer.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/separate_train_mapper.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/separate_test_mapper.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/separate_validation_mapp
https://github.com/ucb-stat-157/Team-2/tree/master/code/group_by_rate_mapper.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/group_by_rate_reducer.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/simple_combine.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/simple_model.py
https://github.com/ucb-stat-157/Team-2/tree/master/code/final_title.txt
https://github.com/ucb-stat-157/Team-2/tree/master/code/final_description.txt
https://github.com/ucb-stat-157/Team-2/tree/master/code/final_purchased.txt