Data Science Workshop - Outbrain Click Prediction

Submitting: Ron Ludmer, Srulik Astrachan

Prologue:

In the following project we have participated in "Outbrain Click Prediction" contest, hosted by Kaggle. The goal of the contest is to predict as accurately as possible, which ads presented to a user will most likely to be clicked by him, using information given in the dataset about the user, the ads, etc.
The dataset contains a sample of users' page views and clicks, as observed on multiple publisher sites in the United States between 14-June-2016 and 28-June-2016.

Each context (i.e. a set of recommendations) is given a display id. Our task is to rank the recommendations in each group by decreasing predicted likelihood of being clicked.

The submission score is evaluated according to the Mean Average Precision @12:

$$MAP@12 = \frac{1}{|U|}\sum_{u=1}^{|U|}\sum_{k=1}^{min(12,n)} P(k)$$

Where |U| is the number of display ids, P(k) is the precision at cutoff k, n is the number of predicted ad ids.

Dataset:

Link to Information about all the tables and the features they contain can be found in bibliography section

Data Preprocessing:

Data preparation: Since the data is split between many tables, first we needed to merge all tables to get the full picture. Train table alone has 87 million rows, and page views table has more than 2 billion rows, so we had no choice but to slice the first and ignore the latter.
We trained our data each time with only a small fraction of the train table (usually 20%).

train-test split - Up until the end of the competition, the train and test table were already separated for us. After the competition ended, we split the train table so that the first 4/5 display id's were in the train and the last 1/5 comprised the test. The reason we did not split them randomly, is to uphold the same relation between the train and the test as in the competition – the test was comprised only from display ids taken from a later time.

Data cleaning: Except for a single table (documents meta), our dataset contains very few nulls and in most tables it even has none, so this step was not needed. Same for outliers – since most features are categorical and their value has no meaning.

Feature Selection: Our goal was to find the features affect the most on the label.
Useful tool commonly used in these situations is PCA (Principal Component Analysis).
PCA is often useful to measure data in terms of its Principal Components rather than on a normal x-y axis. Principal Components are the underlying structure in the data. They are the directions where there is most variance, the directions where the data is most spread out. PCA works best for discrete features. Unfortunately, almost all attributes on our dataset are categorical with unique values spreading on very large range, therefore we could not use PCA.

We selected features by manually reviewing our dataset and extracting information on them.
We started by eliminated "bad" features:
traffic_source – found only in the page views table.

uuid – exists only on events table and page views table and has no meaning by itself, therefore not useful to us.
source_id, publisher_id – features with metadata about some of the documents. Holds information about very small amount of all documents therefore has very high null rates.
publish_time – has high rates of corrupted values (future dates) so can act as noise and decrease score.
document_entity – has very high rate of unique values, therefore offers no pattern which we can learn from and train our data with.

The remaining features were used in our prediction as predictors or fields for feature engineering: { **ad_id, document_id, display_id, category_id, topic_id, confidence_level, platform, geo_location, timestamp, campaign_id, advertiser_id** }

Since our data is almost entirely comprised of categorical fields, it cannot be given as predictors to a ML algorithm. The obvious solution in this situation is to use dummies (one-hot), but since almost all categorical fields had hundreds or thousands of optional values this becomes quite a challenge.

platform - The exception of the previous paragraph is the "platform" field. It has only 3 possible values: desktop (1), mobile (2) and tablet (3). Also, it had almost no NA values (5 in total), which makes it an ideal candidate to add encoded as one-hot.

topics & categories - Our second feature was the documents' topics and categories. At first we tried to one-hot encode all the topics and categories, but since they had too many unique values (97 unique categories, 300 unique topics), computation was impossible.
We tried to solve that problem with binning correlated topics and correlated categories together decreasing their number drastically, but with no success (more on feature engineering).

Despite the fact we could not bin them together, we decided to keep the topics & categories feature, since we had a strong belief that they affect drastically on the label.
We merged both topics & categories tables into one, imputing nulls (document who had only topic or only category) with topic/category -1, and confidence level 0, in order to flag them.

display_documents & ad_documents – The latter feature was also designed to test our assumption that the subject of the document displayed to the user and the subject of the ad are highly correlated. Promoted content table holds connections between ads and the documents they lead to. The downside is that this table is relatively very small, so it contains many nulls once merged.

Feature engineering:

This was the main work of our project. Since the last features of course are not enough for prediction, we had to extract meaningful connections from the data.
Click Through Rate – very common feature in this kind of contests. It acts as a "strength" measurement for a single or group of features. It's based on the simple idea that the higher the features' click rate, the more likely it will be clicked again in the future.
For every feature (and some relevant combinations of them) from the features { **ad_id, document_id, campaign_id, advertiser_id** } we computed their CTR: number of times they appeared in the train with the label click = 1, divided by number of total appearances.
This value was later normalized (see ctr function in ctr_features.py or Bibliography for reference).

Topics & categories correlations – This feature was originated as a way to bin topics and categories together. From the CTR feature we had correlation scores between ads and documents. Given a "strong" ad-document pair, if we know their topics/categories we can transform the CTR score to a correlation score between their topics and categories.

Our original intent was to identify highly correlated pairs, and merge them into one topic/category.
This failed because at the end we were left with very few sets containing each numerous amounts of topics/categories (It happened because some categories/topics were so popular they had high correlation with almost all the rest, and so grouping them all together).
Instead, we decided to create dictionaries with pairs of topics/categories as keys, and correlation "strength" as value.
The "strength" of a pair was their CTR value, multiplied by their confidence levels.
In order to keep only real strong and trusted connections, we used thresholds for confidence level and min score taken.

Timestamp – another feature we believed has great importance is the timestamp, since different hours of the day or the week attracts different audience, which in turn prefers different subjects.
The feature was given in format: milliseconds since 1970/01/01 – 146587679998.
Since we were only interested in the part of the day or the week, we decided to convert it to one hot of 5 possible values: { morning, noon, evening, night, weekend }.
We could not make the transformation as it is, since each timestamp was related to UTC but did not account for the time differences. Therefore, we used several packages to deduce from the geo_location field the relevant offset.

After all features were ready, we merged them with train & test tables. We inspected the rates of nulls for each feature, and discovered that all pair-CTR features (doc_id & ad_id, doc_id & advertisement_id, doc_id & campaign_id, doc_id & ad_doc_id) have very high null values (50% in average). This was expected, since all these features originate from "promoted content" table, which was very small relatively to the rest of the dataset.
This feature could be dramatically improved using the "leak" (more on conclusions).
We imputed those missing values with median values.

Modeling:

Our main focus was on Logistic Regression algorithm, since our research in the matter showed it's the most suitable for predicting on label of True/False.
Also, due to our memory limitations, we opt for algorithms that consider faster, such as Logistic Regression.
Nevertheless, we did try several different models and compared in order to achieve the best score possible.

*We point out that all comparisons were made according to the MAP @12 evaluation score and not the algorithms own score methods. Also, since the competition ended on 18/01/17, and a major part of our progress was made afterwards, all scores presented in the notebook are not scores given by Kaggle, but scores evaluated by computing MAP @12 on our test table.

Feature Selection:
First, we split the train into train and validation tables (same as before - first 80% are train, last 20% are validation).
Before we choose algorithm, we wanted to find the best combination of our features to act as predictors. We did it by implementing kind of a "grid search", by running various combinations of our features on Logistic Regression model (we chose Logistic Regression because it's the fastest and this step takes a very long time).

Model Selection:
The models we compared are: **Random Forest, Gradient Boost, Logistic Regression**
For each model, we ran grid search on 2 chosen parameters we believed to have the greatest effect on the result.
The predictors for each model were the optimal ones chosen the step before.

Prediction:

After we used the validation set to find best predictors and optimal values for all algorithms, we made prediction on the test table for each algorithm, and compared the final results.
For the final prediction, we used 2 different measurements for success:

1. MAP@12 score – same as before
2. Portion Predicted – we calculated the percentage of display ids which we predicted correctly the clicked ad.

Conclusions:

1. As expected, Logistic Regression provided the best prediction scores between the models we tested at most cases.
2. Our prediction scores varied in range from 0.650 to 0.658, depending on our train/test split, the values we used to fill NA values, and the values we tested on the grid search. No "winning combination" found, meaning we used a lot with trial and error.
3. Topics & Categories correlations and our time feature did not prove themselves as helpful. At most runs they were not included in the best combination of features, and most of the time even decreased the final score.
4. Needless to say were not fully content with our results and we expected better. We believe that our difference in score from the other competitors was founded of the following:

- Lacks of experience – we both have no experience in DS. As first timers to encounter such massive and complicated dataset was very challenging, and we did many mistakes at first trying to find the right way to progress.
- Computing limitations – all through the project we had to ignore/slice huge amounts of data, since we could not manipulate it. This drastically damaged our chances to improve our features. Prominent example of that is the "leak" feature – the "leak" feature refers

to the connection between ads and the documents they refer the user to, after being clicked. This is a very strong feature given only in the Promoted Content table for a very small percentage of the ads.

The “leak” is found by looking at a certain row in the Events table containing information about clicked ad, user and timestamp, and then tracking the page this user has seen few seconds later in the Page Views table. From the page in the Page Views table we can infer the associated document with the clicked ad. It’s called a “leak” because it was information that was not intended to be given. In the forums competitors estimated that using the “leak” boosts the score by about 0.15!

- Low number of features – It is known that the more features the better your prediction is expected to be. We engineered a relatively small amount of features, in compare to the other competitors (example in bibliography). This was also due to memory limitations and inability to process huge amounts of data.

Bibliography:

https://www.kaggle.com/c/outbrain-click-prediction/data - dataset information

https://www.kaggle.com/ashhafez/outbrain-click-prediction/evaluate-map-score-without-groupby-in-python - evaluate MAP score

https://www.kaggle.com/anokas/outbrain-click-prediction/regularized-btb-0-635 - CTR example with regularization

https://www.kaggle.com/c/outbrain-click-prediction/discussion/27926 - example from the forum for possible features