ISOM 674 Machine Learning Final Team-Based Project

Team 7

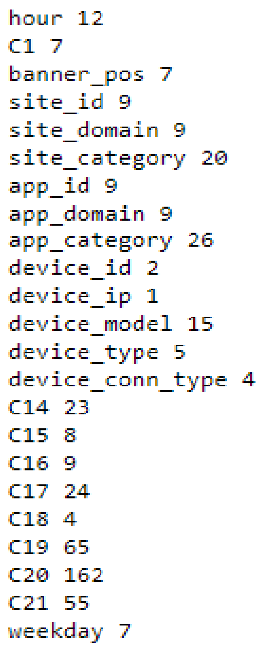
Wei (Will) Jiang, Xiang (Maggie) Meng, Paakhi Srivastava, Yulin Gai, Joe Ebby Karuthedath

1. Introduction & Data Understanding

For our final project, we were assigned an ML prediction problem where our goal is to create a model that correctly classifies if an on-line advertisement will be clicked on. We were given a large data set, with 30 million observations, from 9 days in October 2014 with features such as 'click', 'site_id', 'site_category', 'app_domain', 'device_model', and several anonymized categorical variables.

During our first look at the data it became clear that the challenge of this project will be working with a large amount of categorical data. As some features had thousands of categories, and we were already starting with millions of data points, it was intuitive that transforming all the features to dummy variables would not be practical.

To start working with the data, we randomized the order of the 30 million observations and split the data into 30 files of 1 million observations each by using Linux commands* (attached in the appendix). We loaded a subset of the full dataset into python and conducted exploratory data analysis to better understand the number of categories within each feature and the distribution of data within each feature. We found that a notable number of the 24 features contained several categories with only one observation within them. We deemed that these categories would not be very informative to the model and it would be efficient to group all categories with less than 100 observations into an 'Other' category for the pertaining variable. This transformation greatly improved our dataset as we went from features containing thousands of categories to ranging from 4 - 162 categories. During this process, we noticed one of the features, device_ip, was transformed to only contain one category which was ‘Other’. This tells us that every observation in the dataset had a different device_ip. The following figure shows the unique values in each category.



```
hour 12
C1 7
banner_pos 7
site_id 9
site_domain 9
site_category 20
app_id 9
app_domain 9
app_category 26
device_id 2
device_ip 1
device_model 15
device_type 5
device_conn_type 4
C14 23
C15 8
C16 9
C17 24
C18 4
C19 65
C20 162
C21 55
weekday 7
```

Figure 1. Unique values in each category in a sample dataset

By the end of our project, our modeling process underwent three phases, within which we attempted to improve our model. Each phase contained its own data preparation and feature engineering process as we continuously altered the data format to best fit the model we were building in that phase. Our first phase consisted of selecting features that would produce the minimum amount of dummy variables and building models strictly from the selected features. In order to include the rest of the features for our second phase, we considered MFA (Multiple Factor Analysis), which is similar to dealing numeric features with PCA. Finally, we used Label encoding and fit in the Neural Network.

2. Methods that failed

Since each column contains a huge number of unique categories, it is impossible to perform the One-Hot Encoding method and directly use the pd.get_dummies() function for each column. Instead, we need to use some other encoding methods to prepare our dataset for modeling. The first method we use to do feature engineering is the count/frequency encoding. Before that, we first transform 'hour' to 'weekday' and 'hour'. We extract weekdays from the original date as a new column 'weekday', we also extract hours from the original column, on the top of that, we split the hours as 6 hour cuts per day (4 hours in each cut) as our new 'hour' column. After that, we used a count/frequency encoding function to calculate the count of each unique category within each column. For example, if we apply the function to the column 'site_id', it will result in the total count of each corresponding category. Since all of our dependent variables are categorical variables, we can apply this function to all the columns. Next, we normalized all the numbers by dividing them to the largest count in each column, which can give us a number from 0 to 1 in each column (that is frequency). The next step is to balance and to make the numbers of 0's and 1's in the 'click' column the same. After that, we split the train and test sets from the data frame. Since all the values are numerical, we apply a logistic regression based on the cleaned data. Unfortunately, the log loss score performs badly, ending up around 0.66. This is a non-informative value since it is greater than 0.5, meaning that a random guessing is even better than this method, so we decide to try another way.

Then we tried the MFA(Multiple Factor Analysis), which is similar to dealing numeric features with PCA, by using the prince package. More specifically, MFA is meant to be used when we have groups of variables. In practice it builds a PCA on each group depending on the types of the group's variables, and it then constructs a global PCA on the results of the so-called partial PCAs. We found this method and detailed package information on Github. Since it’s our first time to use it and we also need to generate dummy variables first for the column, we decide to choose two columns, namely C21 and C20 to test this method and see if it is useful for this dataset. We generate dummy variables by using One Hot Encoding for C21 and C20 columns firstly and assign them to two dictionary groups. After the encoding, there are 215 columns in total. Then we perform the MFA to those 2 columns. However, when we check how much information the result can explain, we found that the percent rate is really low (0.0084 and 0.0079 respectively). Thus, since we lost a lot of information after the data transformation, it seems that MFA is not suitable in this case, so we decided to move on and try other transformation methods.

3. Method that works

After trying different kinds of transformation methods and models as well, we thought maybe the Neural Network is suitable for this kind of huge data set. One advantage about the Neural Network is that it can “learn” to fit the categorical data by itself so there is no need to do much complex feature engineering before modeling. Hence, we dropped some columns that are less informative and transferred categories that have less than 100 rows to “others” since those categories are not representative for the click rate. Then we used Label Encoding to transfer all the categorical data to different numbers that represent unique categories and started to build our Neural Network.

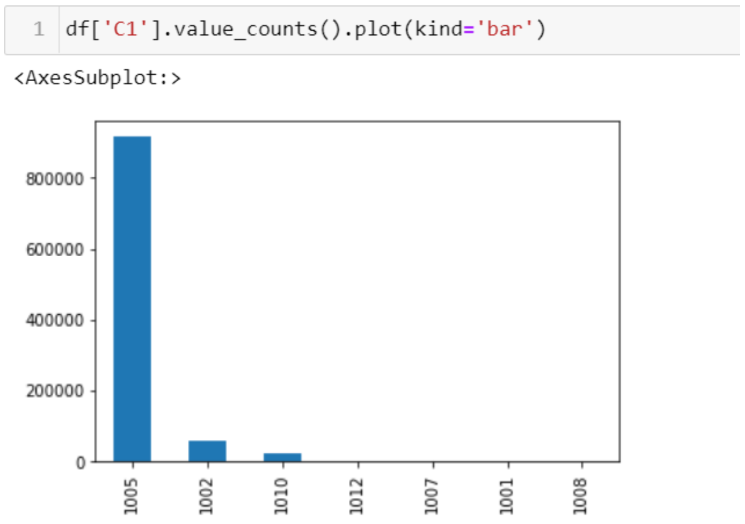
3.1 Data preparation

3.1.1 Splitting the Training Data

The training data was split into subsets of 1 million each for improving the workability of the data set. This was done using the linux command line interface.

3.1.2 EDA

A few categories tend to dominate the dataset; an example can be observed above where the category 1005 dominante over the other variables in feature C1. This is the case with most of the features where one category dominates over the others!



```
df['C1'].value_counts().plot(kind='bar')
```

```
<AxesSubplot:>
```

Figure 2. Example: a few categories tend to dominate the dataset

3.1.3 Data Transformation

The datasets had a very categorical nature. An exceeding number of categories had less than 100 instances(rows). The device_ip, id and hour columns were dropped. Device_ip, id did not make a lot of sense in the model as most of them are unique to a device. The hour column crashed the model a few times, which gave the team a good reason for it to be dropped.

Since we are using a Neural Net the categoricals were converted to labels using Label Encoding and the non linearity of the model was left to account for the categorical data.

3.2 Modeling

For modeling, after trying several different methods, we finally decided to use Neural Network to handle such a large dataset. But as we mentioned before, all of the predictors in the dataset are categorical variables, which need to be transformed to feed into the model. First, we tried to cut all the categorical data into bins based on their frequency and CTR; however it did not work well, since much information was lost during that process. Also, for some of the predictors, there are thousands of different categories, which means that if we use One-Hot Encoding, it'll generate thousands of columns. To solve that problem, we finally decided to use Embedding Layer and Neural Network.

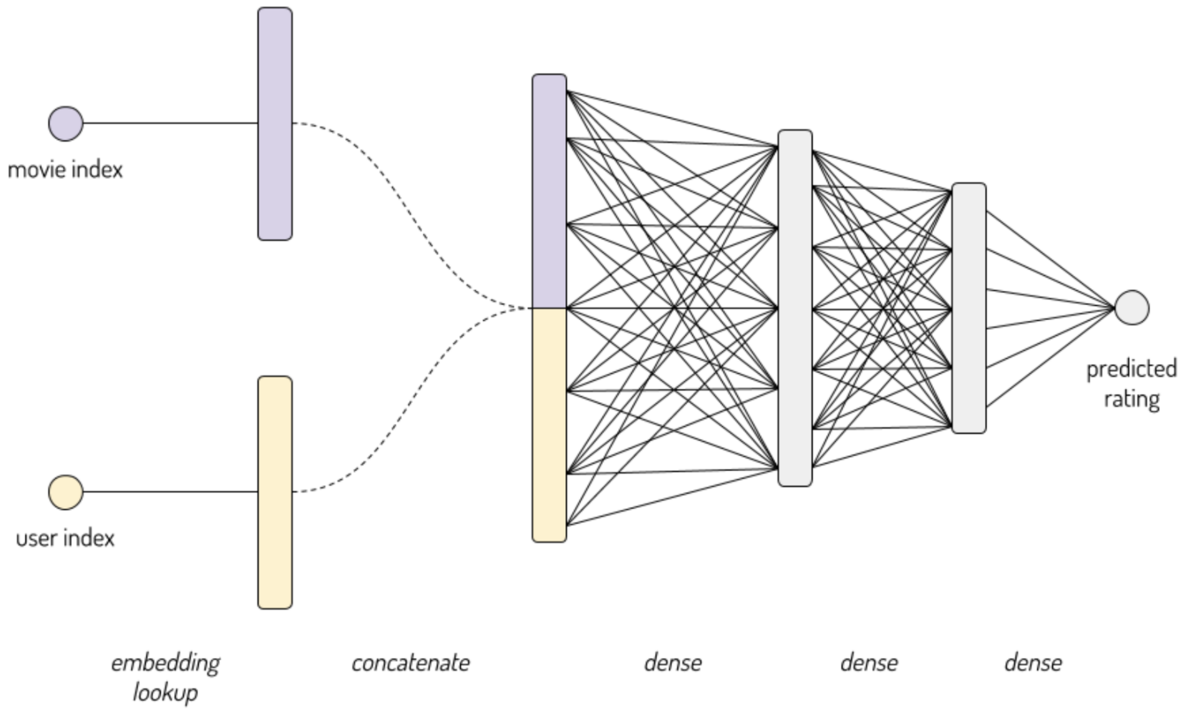


Figure 3. Embedding Layer and Neural Network Example

To decide the dimension of the embedding layers, we chose a more flexible threshold – if a half of the number of categories is still larger than 30, then we'll use 30; and a half of the number of categories otherwise. Through the embedding layer, we transform that originally tens of thousands dimension space into a much smaller one – 30 at most for each of the predictors. Then we implement a deep neural network with 4 fully connected hidden layers and 4 drop-out layers to prevent overfitting. The first two of them include 256 neurons respectively, followed by one with 128 neurons and another one with 100 neurons. The activation function is Rectified Linear Unit (ReLU). Besides, the final output layer has only one neuron with activation function as Sigmoid.

So in our model, we have around 380,000 parameters and we set the max epoch as 1000, batch size as 512, and use "Adam" as optimizer. Also we set the early stopping tolerance at 15.

3.3 Evaluation

We use Log Loss function to evaluate the performance of our model prediction, the function of Log Loss is shown below:

$$Log\ Loss = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log p_i + (1 - y_i)\log(1 - p_i))$$

p_i stands for the probability that our model attributes to the real class, and y_i stands for the actual value ('click' in our dataset). The log-loss function indicates how close our prediction probability is to the corresponding actual value when our target variable is binary. Therefore, a lower log-loss score means our prediction is more accurate. Our prediction turns out to have a mean log-loss value around 0.39 based on several validation dataset we choose, which is a huge improvement compared to the results we got from using other approaches. Thus, our final model appears to have a stronger predicting ability towards whether users click the advertisement or not.

4. Appendix

Frequency_encoding.ipynb: EDA plots about the dataset; using frequency encoding for feature engineering; using logistic regression and XGBoost for the prediction model.

Basic_Logistic.py: This code was an attempt to run the most basic Logistic regression model for making the prediction, which then can be kept as an insurance if things go south. Only the features with a reasonable number of categories were kept and the others were dropped. These were then converted to dummy variables. The logistic regression model gave an accuracy score of 83%. The Logistic regression was further optimized using hyper parameter tuning, by using log loss as the scoring parameter. This gave us a log loss of 0.43, which was reasonable. But our pursuit to build a better model continued.

MFA.ipynb: Attempt to use MFA(Multiple Factor Analysis) to perform dimension reduction.

Final_Neural_Net: Our final model by using embedding layers and Neural Network. We use the model to predict the test data for submission.

Linux commands for shuffling and splitting the data:

```
shuf ProjectTrainingData.csv >  TrainPerm.csv
split –l 1000000 –d –a 3 --additional-suffix=.csv TrainPerm.csv Train-
head -n 1 test_000.csv > ProjectSubmission-Team7.csv && tail -n+2 -q *.csv >> ProjectSubmission-Team7.csv
```