ML ORFE Kaggle CTR prediction

Team L&Y (Aeri Liu al4109, Doohyung Yun dy2423)

EDA:
The nature of the dataset, most of the variables being non-ordered categorical and even many of them anonymous, renders common approaches of EDA quite meaningless. Yet, we were able to apply pandas groupby functions to visualize some of the relationship between independent variables such as id, ip, etc. , with click variable. This way, we discovered few variables had too little variability and were good to be removed from analysis during feature engineering. Overall, the data displayed a highly nonlinear pattern and multimodal. Thus either state-of-the-art classifier libraries or neural net were thought to be necessary to reach the target prediction score.

Method for final submission:
Catboost was used for our final submission. Other than using the original features alone, we added an auxiliary feature called ‘int_hour’ which extracted the real hour in integer from the ‘hour’ column. Used columns of object type as categorical features and all other features including id was considered numerical. After the first try, according to the feature importance function supported by the package, we adjusted the features and tuned parameters for 4 times and finally achieved the optimal result. We increased the number of iterations to get the best possible prediction but at the same time used overfitting detection settings to avoid overfitting of the training set. As a result, we could get log-loss of 0.40607 for our validation set.

Other Attempts:
We first tried light gradient boosting. Although we tried normalization of the data, over 50 times of parameter tuning, adding and deleting features, considering features as categorical and numerical, we couldn’t get below 0.45298 log-loss score. So we thought our method for encoding the label had a problem (we used label-encoding) and changed our method to using catboost which encodes the categorical features by the model itself.
Some of the methods used beside boosting techniques are neural net and random forest classifiers. Neural net was implemented with PyTorch, with a simple MLP architecture and backpropagation. We had to convert scikit learn csr_matrices to torch.tensors by referring to https://ray075hl.github.io/ray075hl.github.io/sparse_matrix_pytorch/. Since ordinary sklearn one hot encoder produced different column sizes for train, validation and test set, using the pkl method was necessary. After converting the sparse csr_matrices to torch.tensors, we tested multiple variations of the neural net, varying in optimization algorithm method, hidden layer structure, hidden activation functions. Among many activation functions, considering the size of the data and the time it took for the local computing environment, intermediate activation functions were set to be ReLU, and the final activation function was a Sigmoid so that the output becomes a scalar probability value.
Despite 80+ different runs for parameter tuning, our model was not able to get below 0.46 log-loss score.
Random forest classifier was implemented by using sklearn. After label encoding the pd.DataFrame type data, we normalized it using sklearn.preprocessing. We tried hyperparameter tunings with the tree but most of the attempts kept us at confusion matrix accuracy at about 75-80%.

We speculate that the under-performing of neural net is due to the data; most variables are categorical without order, and produces an extremely sparse matrix when one-hot encoded. Such data structure doesn't give enough information to the neural net, making classifiers perform comparatively better.

Resources consulted:

- Official documentation for respective packages used (sklearn, pytorch, catboost, light gradient boosting)
- Codes from the Avazu CTR Prediction challenge