Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Airbnb New User Booking Challenge: an Ensembling-based Approach

Authors:
Giulia Chiaretti - 800928- g.chiaretti@campus.unimib.it
Federica Fiorentini - 807124 - f.fiorentini1@campus.unimib.it
Riccardo Maganza - 808053 - r.maganza@campus.unimib.it

February 4, 2020


UNIVERSITA' DEGLI STUDI DI MILANO
BICOCCA

Abstract

The Kaggle Airbnb New User Bookings Challenge is tackled through a solid ensembling procedure combining gradient boosting machines and neural networks. Its result is encouraging but shows that more care with the resampling procedures and hyperparameter optimization is needed to achieve better scores.

1 Introduction

The challenge tackled in this project was the *Airbnb New User Booking* prediction challenge, a Kaggle[1] competition launched in 2015 by the Airbnb Data Science team to recruit prospective Data Scientist and Data Analysts for their research team.

The goal is to correctly predict where to, if any, a user will book their next stay through the platform among 12 possible options. The problem is thus a *multiclass classification task*.

The evaluation metric for the task was the *Normalized discounted cumulative gain*, i.e. NDCG, the definition of which can be found on the competition page[2]. The metric compares an output ranking of the possible outcomes, to the ground-truth value, and assigns higher values if the true output is ranked higher by the model. This competition in particular only considered the first 5 predictions as ranked by the model, hence the name of the metric *NDCG@5*. The single NDCG values are then averaged to obtain the submission score.

The problem is thus a *Learning to Rank*, LTR, task as well, which can lead to some different intricacies than the ones of a regular classification task [1], some of which will be discussed later.

The available data consists of information extracted from users' profiles and of some session data.
Different data came from different sources, each provided in their file and not always referring to the same time interval. This led to many problems regarding the handling of missing values and initially shifted the focus of the work to data preparation, feature engineering and feature selection.

[1] `https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings`

[2] `https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings/overview/evaluation`

The task was also an unbalanced one, with more than 80% of the target variable's training set belonging to just 2 of the 12 classes. For this reason, oversampling and undersampling techniques were also analyzed, leveraging the trade-off between improving the data quality for the subsequent modeling and the computational power and time requested by the various techniques.

After a long data preparation process, the task was not deemed big enough for a regular neural network to tackle it: the small size of it would have surely lead to overfitting without the application of important regularization techniques.

For this reason, as a further means of regularization, two ensembling meta-models were built and implemented as the solution of two different, but related, mathematical programming problems.

By making use of the authors' previous experience in Kaggle challenges, *gradient boosting* models were also used as input for the ensembling procedures, as they had always proved to be a good starting point for this kind of classification tasks.

Finally, two parallel SMBO-based approaches were used to select the best set of hyperparameters for the MLP and the gradient boosting models.

2 Datasets

Original data was provided through five different *csv* files, for the full list of files and variables, see here. Only the *train_users2*, *test_users* and *sessions* datasets were used, as the other two were deemed not informative. The first problem was that the three datasets referred to different periods. This meant that a lot of information would be lost, regarding users present in the training set but with no recorded session data. Only about half of the training set had some corresponding session data.

The feature engineering procedure was built disregarding redundancy, with the thought of removing potentially redundant variables within the feature selection procedure.

Starting with the sessions data, the following information has been added:

- For the *action*, *action_type*, *action_detail* and *device_type* the counts of each value for each user in addition to the sum and standard deviation

of thouse counts;

- For the *secs_elapsed* variable, the logarithmic transformation of its sum, mean, standard deviation and median for each user, in addition to the number of times very long or short breaks were present. The logarithmic transformation of the variable was also split into 10 equally-sized bins and for each user, a variable accounting for the number of the values in each bin was added. The missing values were imputed as the mean seconds for each user, if available, or with the global mean.

After concatenating the train and test datasets to unify their preprocessing, the following variables were also constructed:

- A variable denoting the number of null values in the user's profile fields.
- The age variable was imputed and recoded in different ways according to its value, which was sometimes clearly a human error and due to a lack of input validation in the website frontend, for example, someone claiming to be 1985 year old. After recoding it, it was split into 20 equally sized bins and treated as a categorical variable.
- The raw Unix timestamp was extracted from the *datetime* variable, as well as dummies representing day of week, month, and hour (when available). One more extracted feature was the difference between the date of the account creation and the date of the first activity.

After joining the resulting dataset with the session data one last feature was created: a dummy representing whether there was session data available for the user.
The rest of the categorical variables were one-hot encoded and the missing values filled with the constant value -1, which was out of the bounds of all the variables and therefore would not pose a threat for the modeling.

The procedure resulted in 652 features.

The data was then split back according to the original training and test splits, and 25% of the labeled training data was reserved for model validation.

Furthermore, after some initial explorations and as already mentioned, it

was clear that the task at hand was very unbalanced, with more than 80% of the training set's target variables having either the *NDF* or *US* outcomes. The cause of this is probably a very localized dataset, and that impression is also confirmed by the *language* variable: 96% of the users in the training set have English set as their *locale*.

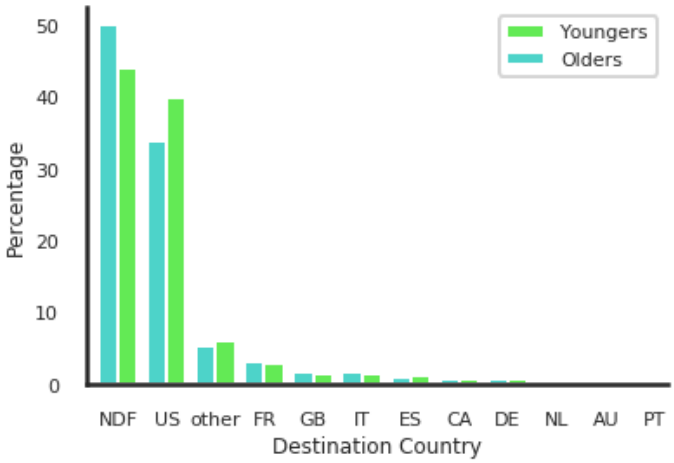

Figure 1: Target Variable by Age

The oversampling and undersampling techniques adopted to overcome this issue will be analyzed in the next section.

3 The Methodological Approach

3.1 Resampling the training set

For what concerns the class imbalance, initially the SMOTE-NC [2] algorithm was applied by itself to oversample the minority classes. This unfortunately led to a dataset containing more than 1,000,000 rows, which combined with the great number of features made the dataset very hard to handle computationally and the software even threw an error when trying to persist it to a file.

Eventually, a reduced version of the SMOTE-NC algorithm together with a *"cleaning"* procedure was applied to increase the minority class without making the dataset size explode: each minority class was increased up to

a third of the size of the majority class by synthetically creating samples through the SMOTE-NC algorithm, then the *Edited Nearest Neighbours* [3] algorithm was applied to *clean* the generated synthetic data by deleting observations which class labels did not agree with the label of at least half (rounded up) of its $k = 3$ nearest neighbors.
The procedure led to a dataset of 49988 samples.

3.2 Feature Selection

The resulting dataset still had a lot of features, many of which redundant. Therefore a quick and easy to implement method for feature selection needed to be found.
Two wrapper models for feature selection were chosen and implemented: Random Forest and XGBoost, which both had analogies with the XGBoost classifiers applied in the following phase and one big advantage: being unaffected by multicollinearity.
The models used as feature selection wrappers were not optimized as they were only used for their feature importance rankings. The top 3/4 of the features (489) were selected, while the rest was discarded. The threshold was arbitrary and empirically set.

Eventually, only the XGBoost wrapper was used in the pipeline for lack of time. The wrapper managed to remove redundancy with minimal loss of information.

3.3 MLP

For the MLP part of the ensembling process, the architecture was first built by fitting a 3-hidden layer neural network with, respectively 256, 128, and 64 layers to the original full training data and ReLU activations for the hidden layers.

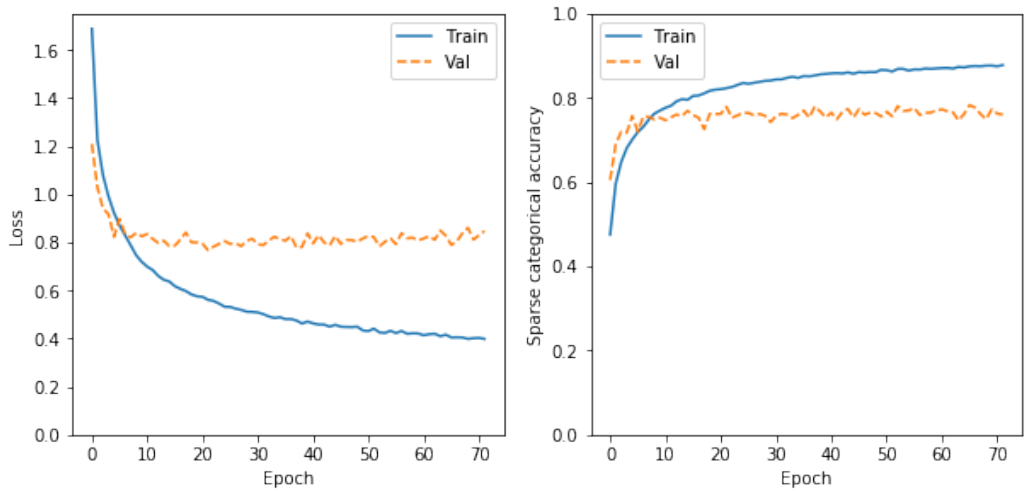

Figure 2: Training and validation measures for MLP without regularization

After visualizing a quite strong overfitting Dropouts were added, ReLUs were changed to ELU and, finally, L2 regularization was added to the first two hidden layers.

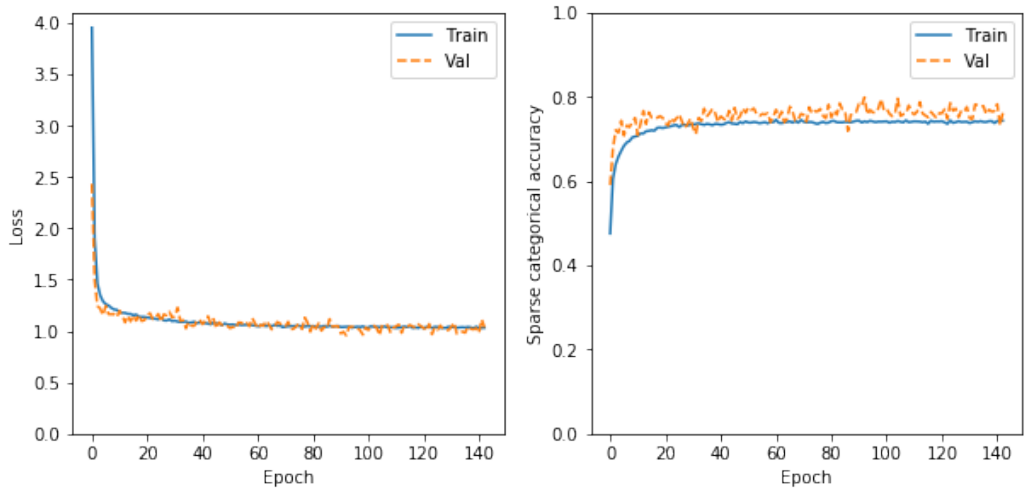

Figure 3: Training and validation measures for MLP with L2 regularization

The overfitting phenomenon decreased and so this final architecture was selected as one of the building blocks of the ensembling procedure.

3.4 XGBoost

XGBoost is a powerful cross-language implementation of a gradient boosting algorithm, which someone calls the *Holy Grail* of machine learning competitions [4]. It is usually very robust and quick but still allows for fine-tuning via its hyperparameters.

For these reasons, and its good initial unoptimized score already obtained in the feature selection procedure, *XGBoost* was selected as the second building block of the ensembling procedure with an architecture similar to the one adopted in the feature selection step, with the addition of an L2 penalty.
Furthermore, the decaying learning rate has been implemented, as it was seen to be beneficial for the model.

3.5 Ensembling Procedure

The XGBoost and the MLP models are each fit on N subsets of the original training data sampled with replacement to further help with regularization, thus obtaining $K = 2N$ set of predictions.

Then, two different mathematical programming problems are set up to aggregate the estimates:

$$\begin{aligned} \underset{w}{\text{argmin}} \quad & \sum_{n} \mathbb{L}\left(y^{(n)}, \sum_{i=1}^{K}\sum_{j=1}^{12} w_{ij} X_{ij}^{(n)}\right) \\ \text{s.t.} \quad & 0 \leq w_{ij} \leq 1 \quad \forall i, j \\ & \sum_{j} w_{ij} = 1 \quad \forall i \end{aligned} \tag{1}$$

where n denotes the specific sample, $\mathbb{L}$ is the standard log-loss, y is the true label, i denotes the model, j denotes the class, and X is the matrix of probability estimates for all the models, so that $X_{ij}^{(n)}$ equals the probability predicted for the class j for sample n by model i.
And a simplified version of it:

$$\begin{aligned} \underset{w}{\text{argmin}} \quad & \sum_{n} \mathbb{L}\left(y^{(n)}, \sum_{i=1}^{K}\sum_{j=1}^{12} w_i X_{ij}^{(n)}\right) \\ \text{s.t.} \quad & 0 \leq w_i \leq 1 \quad \forall i \\ & \sum_{i} w_i = 1 \end{aligned} \tag{2}$$

In both cases, the goal is to find the best linear combination of the predictions such that the log-loss is minimized. In the first case, there is a separate weight for each class within each model; in the second one weights are shared across the classes within each model.

The problems are solved iteratively through a Newton, or quasi-Newton, solver.

The ensembling procedure is implemented in the `EnsembleClassifier` and `EnsembleSingleModelsClassifier` sibling classes (in the modeling notebook), which are built to be fully compatible with the *sklearn* API and therefore with its helpers, such as cross-validation procedures.

3.6 Calibration

Most classifiers, especially neural networks, are very good at predicting the correct class when minimizing log-loss objective functions, but in LTR tasks the goal is to correctly output ordered probabilities [5]. The need for calibrating probabilities stems directly from the nature of the task. The most promising estimators from the ensembling procedure are calibrated by fitting an isotonic regression to the validation data via the `CalibratedClassifierCV` class and are then used to predict the original unlabeled test data.

3.7 Hyperparameter Optimization

The chosen hyperparameters to be optimized are the ones the models appeared the most sensitive to while fitting, namely:

- MLP
  - the number of the neurons in the hidden layers;
  - the λ parameter for L2 regularization;

  - the learning rate of the Adam optimizer

- XGB
  - the bounds of the decaying learning rate;
  - the learning rate decay rate;
  - the percentage of data randomly sampled with replacement to fit each decision tree;
  - the λ parameter for L2 regularization

The *ensembling* classes have been built with a simple flag that enables HPO with a reasonable set of parameters: a Random Forest surrogate model for the MLP and a Gaussian Process for the XGB since the latter only has real-valued hyperparameters to be optimized. For both of them, the *Lower Confidence Bound* acquisition function was chosen. The chosen objective function is the mean *NDCG@5* across all the bootstrap iterations for each model configuration.

Unfortunately, due to the computationally expensive and iterative nature of the modeling, the HPO results are not very good.

4 Results and Evaluation

All the following results are *NDCG@5* scores for the respective models on the marked data subset. **NB**: All test scores come from Kaggle late submission scores from classifiers fitted on the training set and calibrated on the validation set.

	Validation	**Test**
Feature Selection XGBoost	0.8622	
Ensemble Block XGBoost	0.859	
Ensemble Block MLP	0.871	0.84
Ensemble Problem #1	0.8351	0.852
Ensemble Problem #2	0.72	

Table 1: Model Scores

For what concerns the hyperparameter optimization, unfortunately, every attempt to run the HPO module of the ensembling class resulted in Google Colab crashing after some time and so we are unable to show results.

5 Discussion

The results from the XGBoost classifiers truly affirm its status as a very good out-of-the-box classifier, even with minimal preprocessing and tuning, it can reach scores rivaling other classifiers.

The MLP shows good results as well, so it can safely be affirmed that the ensembling approach was at least promising on paper.

The resampling procedure, which in theory should have helped with generalization, turned out to be detrimental for the model. Perhaps resampling in addition to synthetically oversampling the original data altered its structure too much and this was noticeable when shifting to never-seen-before data.

Nevertheless, the models are still relatively far from Kaggle's private leaderboard top scores, which reach up to 0.8869.
The authors are positive that with more fine-tuning (including hyperparameter optimization) and more attention on the resampling, better results could be achieved.

6 Conclusions

The goal of this work was building and implementing a complete ensembling pipeline of neural networks and state-of-the-art gradient boosting models, which was surely reached.

With a little more attention, and maybe computational power, some more sophisticated oversampling/undersampling/weighting techniques could have been developed, which would have led to better results when combined with the ensembling pipeline which could have gotten better scores in this specific task.

References

[1] H. Li, "A short introduction to learning to rank," *IEICE TRANSACTIONS on Information and Systems*, vol. 94, no. 10, pp. 1854–1862, 2011.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[3] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 408–421, 1972.

[4] D. Nielsen, "Tree boosting with xgboost-why does xgboost win "every" machine learning competition?" Master's thesis, NTNU, 2016.

[5] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1321–1330.