Défi IA 2023
1001 nights: prédire le prix d'une réservation d'hôtel

Mohammad Badia Alfathi [1] **Mostafa Hajj Chehade** [1] **Beltrán Castro Gómez** [1]

Abstract

This document aims to present our team's participation and the approaches we followed during the *Défi IA 2023* Kaggle Competition (ElM et al., 2022), which involved several students from french and francophone universities, grouped into a total of 76 teams of at most 3 members each.

1. Introduction

The *Défi IA 2023* Kaggle Competition consisted in hotel room price prediction, with the peculiarity of the absence of training data. Initially, we were provided just testing data and hotel metadata and were given three tasks:

1. Collect our own training data through an API that simulated a hotel recommender system to which we had restricted access in the form of 1000 queries per 7 days on a sliding window of time.
2. Find the bias in the recommender.
3. Perform regression on the hotel rooms price.

The recommender API request parameters were $[city, date, language, mobile, avatar]$ and its response included a list with hotel IDs, prices and remaining stock.

2. Exploratory Data Analysis

In the first instance, we performed some exploratory data analysis on the test data with the goal of understanding a bit better the challenge before going ahead with the collection of the training data. By doing so, we made one main discovery: the testing data was clearly imbalanced.

On the one hand, we could observe that the amount of *one-time requestors*, or users that submitted only one request, was huge compared to the rest. The maximum number of requests submitted by one same user was 4, as it can be observed in Table 1.

Request count	User count	User Percentage
1 request	749 users	95.41%
2 requests	17 users	2.17%
3 requests	15 users	1.91%
4 requests	4 users	0.51%

Table 1. Distribution of number of requests performed by users in the testing set.

On the other hand, we also found imbalance on some of the API parameters. For the *city* parameter, we observed that cities like Paris or Amsterdam are each present in around a 17% of the test examples, while other ones like Valletta were present in only a 4.5%. For the *language*, the hungarian language was by far the most popular one, appearing in a 15% of the samples, whereas the portuguese language, for example, was only in a 1%. Lastly, for the *date* parameter, we could see how not only 24 of the possible 45 dates were not even present in the testing set, but also that dates lying in the range $[0, 6]$ were found in almost the half of the testing data.

3. Data Collection

In this section, we will do a chronological overview of the four approaches we followed for collecting training data through the challenge's API.
In the early stages of the competition, we decided to perform some randomly generated queries, with the goal of gathering enough training data so as to draw an initial general idea of its structure and distribution.

Nextly, we went on to study the bias behind the recommender's suggestions, particularly behind the *language* and *mobile* parameters. For that purpose, we submitted batches of randomly generated queries for which we purposely edited the parameters of study to include all its possible values and used different avatars in all queries

*Equal contribution [1]Jean Monnet University, Saint-Étienne, France. Correspondence to: Mohammad Badia Alfathi <mohammad.badia.alfathi@etu.univ-st-etienne.fr>, Mostafa Hajj Chehade <mostafa.hajj.chehade@etu.univ-st-etienne.fr>, Beltrán Castro Gómez <beltran.castro.gomez@etu.univ-st-etienne.fr>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

to avoid injecting user-related biases. This means, that for the study of *mobile*, if our query was $\{"date" = 6, "language" = "greek", "city" = "rome"\}$, we would submit two queries corresponding to the two possible *mobile* values - 0 or 1. For the study of *language*, we would do the same but including all possible 27 languages.
Our experiment found the *mobile* parameter to be free from bias, contrarily to the *language*, which made the price go higher for languages such as finnish or swedish, and lowered the same for languages like slovakian or hungarian, showing up to 14 points of difference between them.

The third strategy we made use of for the data collection was imitating the test set. The main objective of this approach was trying to reduce the divergence between our current train set and the test set, to see if that could improve our regression scores. We did it by adding slight modifications to already existing queries extracted from the test set. For example, changing the *mobile* value from 0 to 1 or adding a one-day difference to the date.
Even if it seemed a good idea, this strategy did not retrieve the expected results, since this approach had probably been anticipated and thus prevented from the challenge creators.

Lastly, we decided to resume our bias search. This time, we analyzed the impact of a user's search history on the received hotel recommendations. We compared the prices recommended to one-time requestors with the ones received by users doing the same queries in terms of *city*, *language* and *mobile*, but with an existing search background. For example, if both users 1 and 2 wanted to travel to Paris and made use of the recommender in their phones and in the german language, but user 1 had already made that query on the dates 40, 25 and 17, at the moment of performing the query on the date 6, our intuition was that the first one should receive a higher price due to its interest on travelling to Paris.
Our assumption was right, nonetheless, this price rise was really hard to track and presumably did not follow any kind of tractable rule or distribution and thus was difficult to measure.

4. Feature Engineering

Informative features make the best out of our models, and that is why feature engineering is very popular among data science and machine learning challenges. It can help us determine which features are the most important, encode already existing variables, create new features and produce segments within our data. Its main goal is to make our data as suitable as possible to the task we are taking.
In this section, we cover the main methods we applied in this challenge to try to improve our scores.

4.1. Subspace Alignment

This method aims to project the labeled source into the target. The target could be unlabeled or could have some labeled and unlabeled data. In Subspace Alignment, both source and target are projected in two subspaces spanned their principal components, to minimize the divergence between the two domains.
We compute X_s and X_t, which represent the d principal components that have the highest variance, of source and target respectively. The alignment matrix M is the matrix multiplication of X_s^T and X_t.
The aligned domain (S_a) of S is the multiplication of the source domain, with X_s and M. S_a is aligned with a projection of T on d-dimension, which means $T_a = TX_t$.
After getting new datasets S_a and T_a, we trained a CatBoost model (Prokhorenkova et al., 2017) using S_a, and predicted the labels of T_a.

4.2. Merging of categorical values

To improve the performance of our predictive model, we considered the possibility of merging categorical values from certain features. This involves grouping together two or more classes from a categorical feature that have similar distributions.
For example, if two classes of a feature, such as *Ibis* and *8 premium* brands hotels, have very similar average prices, we could merge them into a single class *Ibis/8 premium* brand hotels. This helps to reduce the number of classes and simplify the model, making it easier to train and more efficient.
However, in our case, after analyzing the data and the distributions of the classes and applying the models, we found that the distributions of the classes for each feature were distinct and merging them would not have improved the performance of the model.
Therefore, we decided to keep the categorical values as they were, with no merging of classes. This approach allowed us to maintain the integrity of the data and ensure that our model was able to capture all the important information and variations in the prices.

4.3. Modeling hotel availability

We tried to extend the information provided by the *stock* by creating some new features, among which we find worth mentioning:

- The median stock by city.
- The total stock for a given date in a particular city.
- The number of full hotels, i.e. with $stock = 0$, on a given date and city.
- The total number of hotels by city.

4.4. Encoding of user history

We attempted to encode and include as a feature the user history that we described in Section 3, for the models to be able to distinguish the different kind of users and their tendencies.
What we did was simply adding a count for the requests submitted by each user, and assign the count value from each user request to all of the prices corresponding to it.

4.5. Clustering and PCA

Principal Component Analysis or PCA (F.R.S., 1901) is a very well-known decomposition technique, very good at finding linear relationships in the data.
We took advantage of that by directly including as features the resultant components with the highest explained variance.
Clustering algorithms are used to discover groups of similar points in data. In this context, we thought that it could be useful for discovering groups of users that follow specific behaviors or patterns, or groups of brands of hotels that follow similar price policies under the same situations.
We performed K-Means clustering (Lloyd, 1982) for different values of k and encoded as a feature the cluster labels from the experiments with the best silhouette score (Rousseeuw, 1987).

5. Regression Models

Classification tasks are more popular than regression problems in machine learning, mainly due to the fact that they are simpler to understand and interpret, and also more commonly encountered in real-life situations. However, this does not mean that we cannot find interesting and challenging regression tasks, like the one we discuss here.
In this section, we present an overview of the most relevant regressors we have used during the course of this competition, and their performance on the hotel price prediction.

5.1. Random Forests

Random Forests (Ho, 1995) (Breiman, 2001) for regression is an ensemble method that constructs multiple decision trees at training time and returns the average of the individual trees' outputs as predictions.
We tried this algorithm in the early stages of the competition due to its interpretability, simplicity and quick training times, but quickly dismissed it based on its poor regression results.

5.2. CatBoost

Gradient boosting libraries like CatBoost, LightGBM or XGBoost have recently gained a lot of attention in machine learning competitions. They all use decision trees as their weak learners and train each new tree in order to correct the residual errors of the previous one.
CatBoost (Prokhorenkova et al., 2017) is a gradient boosting library specifically designed to handle categorical features. Its algorithm, called *Ordered Boosting*, builds an ensemble of decision trees that, by training the trees in a specific order, deals with categorical variables without the need for one-hot encoding. This makes the model more efficient and less prone to overfitting, as it reduces the number of features.
During the training process, CatBoost also uses various strategies to optimize its performance, such as feature selection and regularization. It also measures the importance of each feature in the model by training the model multiple times and shuffling different features each time.
After training, the model makes predictions on new data by doing a weighted average of the predictions made by each tree, where each decision tree has a weight determined by the gradient descent algorithm.
After applying grid search to tune the model hyperparameters, the lowest RMSE we obtained with CatBoost was 20.44.

5.3. Multi-layer Perceptron Regressor

A Multi-layer perceptron is a feedforward neural network, where all of its layers is fully connected to the next one. It can model complex, non-linear relationships between input features and output values.
For this challenge, we built an MLP with 64 units in the input layer, three hidden layers with 64, 32, and 16 units activated by the *ReLu* function and one unit in the output layer for predicting the price values. We chose adadelta as the optimizer for the training process.
The best RMSE score achieved by our MLP model was of 19.92.

5.4. Model averaging

Model averaging is a machine learning technique that combines the predictions of multiple models to create a final prediction. This is done in various ways, such as by taking the average of the predictions, weighting the predictions based on the model's performance, or using advanced methods like a stacked generalization (Wolpert, 1992).
Model averaging is often used when multiple models (which in our case are MLP and CatBoost) have been trained to solve the same task, and it can improve the performance of the model ensemble. Averaging the predictions of models that are prone to overfitting or underfitting can produce more robust and accurate predictions.
In this project, we got the average of the two predictions, and we reached our lowest RMSE, which is 19.04.

5.5. Stacked Regression

Stacked generalization for regression (Wolpert, 1992) (Breiman, 1996) is an ensemble method which trains several base models on the input data and then their outputs are used as inputs to a meta model, which is trained to make predictions based on the predictions of the base models.
Finally, the predictions of the stacked regression model are made by combining the predictions of the base models and the ones from the meta model, generally by doing a weighted average of them.
Since this is a computationally expensive algorithm, the grid search we performed to tune the different models' hyper-parameters had to be randomized. The best RMSE score achieved was 21.87 for an stacking of a linear regressor, a Ridge regressor (Hoerl & Kennard, 1970), a Lasso regressor (Tibshirani, 1996), an AdaBoost regressor (Freund & Schapire, 1997), an Extra-trees regressor (Geurts et al., 2006) and a gradient boosting regressor (Friedman, 2001) as base models and a random forests regressor as the meta model.

6. Conclusion

As initially presented in Section 1, this challenge involved three main duties.
We effectively collected our own training data, gathering aroun $600,000$ examples.
We also employed several machine learning models to try to predict the hotel prices, among which we must point up the MLP regressor and CatBoost's performances. Moreover, by combining their predictions with model averaging, we were able to further improve the accuracy of predictions.
Finally, the competition also implied finding the bias in the recommender system, which we tried by performing data analysis on our collected data and by submitting to the API tailored queries with the goal of identifying the most important factors that influence the hotel room prices.
However, we do not consider to have completely succeeded in the last one, since a thorough understanding of the recommender's behaviour and its bias would probably had allowed us to even improve our scores a lot more.

References

Breiman, L. Stacked regressions. *Machine Learning*, 24: 49–64, 1996.

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

ElM, Halford, M., Regulathor, and Sixin. Défi ia 2023, 2022. URL `https://kaggle.com/competitions/defi-ia-2023`.

Freund, Y. and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

F.R.S., K. P. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720. URL `https://doi.org/10.1080/14786440109462720`.

Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Ho, T. K. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pp. 278–282 vol.1, 1995. doi: 10.1109/ICDAR.1995.598994.

Hoerl, A. E. and Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. ISSN 00401706. URL `http://www.jstor.org/stable/1267351`.

Lloyd, S. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. doi: 10.1109/TIT.1982.1056489.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. Catboost: unbiased boosting with categorical features, 2017. URL `https://arxiv.org/abs/1706.09516`.

Rousseeuw, P. J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL `https://www.sciencedirect.com/science/article/pii/0377042787901257`.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL `http://www.jstor.org/stable/2346178`.

Wolpert, D. H. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80023-1. URL `https://www.sciencedirect.com/science/article/pii/S0893608005800231`.