An empirical study of Hyperparameter Tuning and Feature encoding Methods for Tree-based Models on Tabular Data

Adeem Jassani
MSc in Applied Computing
adeem.jassani@mail.utoronto.ca

Anusha Prabhudev
MSc in Applied Computing
anusha.prabhudev@mail.utoronto.ca

Apoorv Dankar
MSc in Applied Computing
apoorv.dankar@mail.utoronto.ca

Naveen Thangavelu
MSc in Applied Computing
naveen.thangavelu@mail.utoronto.ca

Abstract

In the current scenario, we see that Deep Neural Networks have state-of-the-art performance for unstructured data but are outperformed by the ensemble of tree-based models in the most commonly used, heterogeneous tabular data. We analyse the following two aspects of tree-based models: hyperparameter tuning methods and categorical feature encoding methods. Since hyperparameters play a crucial role in the model's performance, the first part of this project aims to experiment with different methods of tuning hyperparameters. We empirically evaluate (a) two variants of the Bayesian Hyperparameter tuning method and (b) the Halving Random Search method and compare these methods with the Random Search baseline. Secondly, since the naive one-hot encoding leads to a curse of dimensionality for high cardinal categorical columns, the second part of this project empirically evaluates various categorical feature encoding methods.

1 Introduction

Hyperparameter tuning plays an important in the machine learning pipeline as it has a major influence on a model's performance and its ability to generalise well. That being said, it is a computationally expensive step as firstly, the search space increases exponentially with the number of hyperparameters and secondly, evaluating each candidate hyperparameter requires training the model from scratch and scoring it on the validation set. Therefore, in this work, we perform an empirical study of various hyperparameter tuning methods and compare them with the popular Random Search baseline in terms of time taken to find the optimal hyperparameter and the performance.

Moreover, effective feature engineering is key to improving the predictive performance of any model. It is no surprise that predominantly all datasets consist of categorical features. However, most of the Machine Learning models are designed for numeric features. Encoding categorical features is thus a crucial aspect of feature engineering that needs to be addressed prior to building any model. Feature encoding is a technique to convert categorical features into continuous numeric values, which can then be used for modeling. A very common challenge in this space is that of handling high cardinality features – categorical features with an increased number of categorical values without any natural ordering, leading to the curse of dimensionality. Therefore, through this work, we analyze different methods of encoding high cardinality categorical features.

For our study, we will be working with datasets from the tabular data benchmark presented in Grinsztajn et al. [2022] and the tree-based models: Random Forest and XGBoost as they generally have the best performance on tabular data.

2 Related work

Tree-based models and tabular data benchmark: Grinsztajn et al. [2022] conducted an empirical investigation into the differing inductive biases of tree-based models and Neural Networks (NNs). In this work, they also come up with a benchmark comprising 45 different tabular datasets.

Algorithms for Hyperparameter Optimisation (HPO): Yu and Zhu [2020], in their survey paper comparing different hyperparameter optimization techniques, present us with a systematic review of HPO approaches. They start with basic algorithms like grid search and random search and state-of-the-art algorithms like Bayesian optimization, Multi-bandit methods, Tree Parzen estimators and Population-based Training methods. Halving Random search also known as successive halving is introduced by [Jamieson and Talwalkar [2016]].

Feature Encoding Algorithms: Popular feature encoding techniques include one-hot encoding, ordinal encoding, binary encoding and feature hashing. Seger [2018] has done a comparative study on a few of these. Cerda and Varoquaux [2022] discuss min-hash encoding for high cardinal string categorical features. Pargent et al. [2022] discusses the superiority of regularized target encoding as compared to traditional encodings.

3 Algorithm Descriptions

3.1 Hyperparameter Optimization Methods

As discussed earlier, hyperparameter tuning plays an important role in the machine-learning pipeline. We can search the hyperparameters naively by manually running multiple combinations of hyperparameters or using an automated hyperparameter tuning method. The two popular hyperparameter search methods are Grid Search and Random Search. In Grid Search, we provide values for each of the hyperparameters and train a model with each combination of the hyperparameters. We finally return the hyperparameters which give the best cross-validation score. In the Random Search method, instead of selecting hyperparameter combinations constrained on a grid, we sample combinations from a given prior distribution and return the sample with the best score. The random search typically performs better than grid search given a fixed budget of evaluations because it is better able to explore the search space compared to grid search which is constrained on a grid [Bergstra and Bengio [2012]]. We, therefore, use Random Search as the baseline in our experiments

In this project, we try to ascertain the effectiveness of other hyperparameter optimization algorithms specifically Bayesian Search and Halving Random Search which are explained as follows.

3.1.1 Bayesian Search

The main idea behind Bayesian hyperparameter optimization is making informed choices for selecting the next combination of hyperparameters to test based on information gathered from previous iterations [Koehrsen]. This way, Bayesian Search tries to reduce the number of iterations or attempts to evaluate the entire model by choosing only those combinations which have a high probability of yielding good performance in optimizing the true objective function. This is achieved by constructing a 'surrogate' function, which is a probability model, of the true objective function. This surrogate function is evaluated as a posterior probability which is proportional to the product of data likelihood and prior we have about the objective function. The surrogate function is much easier and requires less computation to evaluate as compared to the true objective function. After constructing the surrogate function, the algorithms optimize this instead of the true objective function and the optimal sample obtained from this process is used to train this model and evaluate the objective function. Then with the information gained by this sample, the surrogate function is updated as the data likelihood function changes and new optimal point is found to train the model on. This is continued till the maximum iteration or time limit. The variants of the methods comes from the way the surrogate function is evaluated. We worked with two different methods - Gaussian Process Approach and Tree Parzen Estimation.

Gaussian Process Approach Gaussian Process 11, or GP, assumes a multivariate Gaussian distribution and builds a joint probability distribution across the variables. As a result, it may efficiently and effectively summarise a wide range of functions and transition smoothly as the model

is given access to more data. This smooth structure and smooth transition to new functions based on data are desirable properties as we sample the domain.

Tree Parzen Estimation The main difference between the GP approach and the Tree Parzen Estimation or TPE [Bergstra et al. [2011]] , is that the GP tries to model the surrogate function as $p(y|x)$ whereas TPE models $p(x|y)$ by applying the Bayes rule, where y is the loss given by the objective function using the hyperparameters x

The TPE algorithm, models the $p(x|y)$ using the data collected in the previous iterations, into two densities as follows,

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases}$$

where $l(x)$ is the density corresponding to the loss less than the threshold y^*, and $g(x)$ is the density corresponding to the loss greater than or equal to the threshold y^*. The TPE algorithms chooes y^* to be some quantile γ of the observed y values such that $p(y < y^*) = \gamma$.

The above formulation of $p(x|y)$ is used to optimize the Expected Improvement criterion.

$$EI_{y^*}(x) \propto \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1}$$

The above equation suggest that we would like the points x that have high probability under $l(x)$ and low probability under $g(x)$. Intuitively, we would prefer points are in the good distribution, that is, the points whose loss are less than the threshold, than the points in the bad distribution, that is points whose loss are greater than the threshold. On each trail, the TPE algorithm draws many points from the distribution l and evaluates them according to $\frac{g(x)}{l(x)}$ and returns the candidate x^* with the greatest expected improvement, which is then evaluated on the true objective function.

3.1.2 Halving Random Search

Halving Random Search (HRS) [Jamieson and Talwalkar [2016]] is a hyperparameter search strategy which is implemented as an experimental feature in sklearn. The sampling scheme for the hyperparameter candidates for HRS is the same as RS i.e, each candidate is drawn from independent and identical distributions after which these samples are evaluated. However, in contrast to evaluating the set of hyperparameters on the entire dataset as is done in Random Search (RS), HRS starts with evaluating the hyperparameters on a smaller dataset of size "MinResources". Secondly, HRS comprises multiple rounds such that for each round we use "Factor" times the number of data points and the top 1/Factor hyperparameters from the previous round, until we use up all the data points. In the final round when we use all the data points, we return the best-performing hyperparameters as the final result.

3.2 Feature Encoding Methods

While standard techniques like one-hot encoding, where we map the feature to a vector that contains ones and zeroes denoting the presence of the category, can be used to encode features with a small number of possible levels, this approach becomes inefficient as the number of levels increases. For high cardinality features, one-hot encoding increases the dimensions, which also significantly slows down the learning.

Another common technique is label/ordinal encoding, where each category is assigned a value from 1 through N, where N is the number of levels of that particular feature. Mapping the inherent ordinality of such features to a wrong scale can impact the model's performance negatively. Similarly, encoding nominal features using this technique will lead to assuming that the levels are ordered even though they do not naturally present any order.

These methods are the most common ones for feature encoding despite their limitations and inefficiency, as simpler strategies are often favored in practice. Thus, finding strategies that work well on a variety of problems is vital for many applications. Our aim is to study and assess the impact of various feature encoding techniques, as described below, on models' predictive performance [8, 12].

Task: Dataset Model	Random	TPE	GP
Regression: california RandomForest	0.82	**0.83**	0.68
Regression: california XGBoost	0.85	**0.85**	0.72
Regression: house_sales RandomForest	0.87	**0.88**	0.88
Regression: house_sales XGBoost	0.88	0.89	**0.90**
Classification: compass RandomForest	**0.78**	0.77	0.70
Classification: compass XGBoost	0.73	**0.74**	0.71

Table 1: Bayes Search result for Random Forest and XGBoost models. For Regression task the metric is R2 score and for classification task it is accuracy

Rare Label Encoding Rare Label Encoding is the process of encoding categorical variables in such a way that the infrequent levels are grouped together into a new single category. Specifying the minimum frequency a category should have enables us to control the number of levels to be preserved. The grouped levels can then be one-hot encoded that the model can use.

Frequency Encoding Frequency Encoding is an encoding technique that maps each level of the categorical variable to its observed frequency in the training dataset. This can be used for nominal features with no natural ordering. It reduces the number of levels, and the model can best differentiate between levels with dissimilar frequencies.

Hash Encoding Feature Hashing or Hash Encoding is a technique that can be used to transform each feature level into a numeric value based on a hashing function. This technique also enables us to control the number of numeric columns produced by the encoding, which alleviates the problem of high dimensionality.

Target Encoding Target Encoding is used to encode the categorical levels of the features by using the target value. In the case of continuous targets, features are replaced with a blend of the expected value of the target given a particular categorical value and the expected value of the target over all the training data. However, in the case of categorical targets, features are replaced with the posterior probability of the target.

4 Experimental Setup and Results

4.1 Hyperparameter Tuning Methods

4.1.1 Bayesian Search

We experimented the two variants of the Bayes Search over a variety of experimental settings with Random Search as our baseline for comparison and inference. We chose three popular tabular datasets - california, house_sales and compass [A.1] - having both regression and classification tasks to evaluate the performance of these HPO algorithms with train-test split of 80%. The models chosen for the experimentation were Random Forest and XGBoost ensembles. So for each dataset, we ran two ensemble models with three hyperparameter optimization algorithms - Gaussian Process Bayes Search, Tree Parzen Estimator Bayes Search and Random Search - giving us a total of 18 experimental settings (3 datasets $\times$ 2 models $\times$ 3 HPO algorithms). The space of hyperparameters to be searched was adopted from Grinsztajn et al. [2022].

For each of these experimental settings, we ran 3 fold cross-validation run and took the mean of metric values obtained from these cross-validation runs as our true objective function. The metric for the classification task was accuracy and for regression tasks, R^2 score was chosen. To reduce the randomness of our results from these experiments, we repeated each setting 3 times for the XGBoost model and 3 times for RandomForest. All search methodologies were run for 40 iterations i.e., tried 40 different samples from the search space of the hyperparameters. We used TPE implementation provided in the Hyperopt `http://hyperopt.github.io/hyperopt/` and for GP we used the skopt implementation provided here `https://scikit-optimize.github.io/stable/modules/generated/skopt.gp_minimize.html`

We compile the results of these experiments as in Figure 1. We observe that the TPE Bayes Search on average outperform Random Search in most cases and tend to reach the optimal hyperparameter sample in much fewer iterations as evident by the large slope of TPE in initial few iterations.The optimal hyperparameter sample found in TPE also outperform the Random Search baseline as seen in Table 1 The GP based Bayes Search either performs similar to the baseline or slightly worse in some cases.

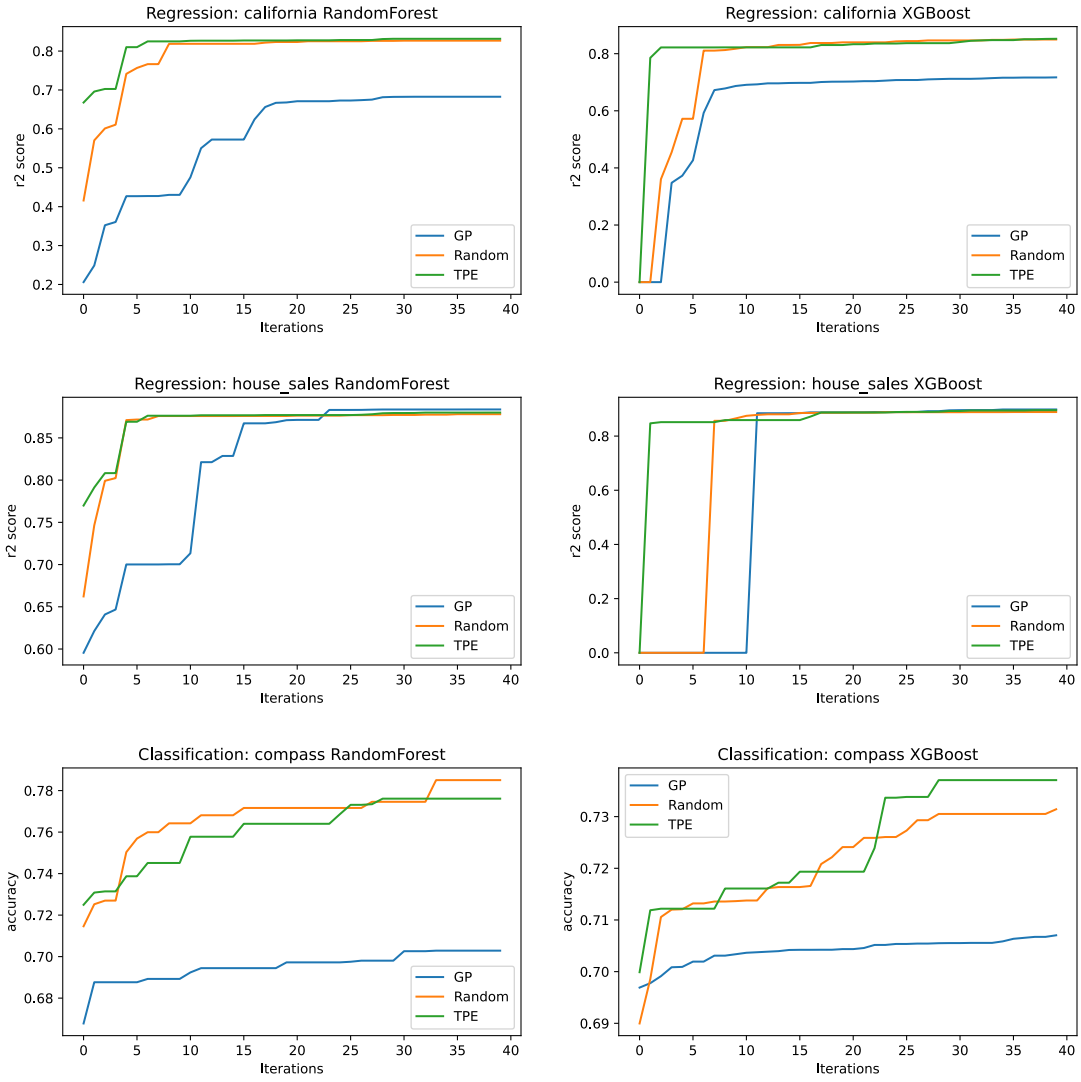

Figure 1: Comparison of GP and TPE based Bayesian Search methods with Random Search over different datasets and ensemble models

4.1.2 Halving Random Search

While HRS may be faster than RS as it starts with a smaller dataset it may or may not converge to the optimal hyperparameters found by RS given both methods have the same starting set of hyperparameter candidates. This is because using smaller dataset results in estimation errors for scoring (cross-validation score) and the best hyperparameter setting may be eliminated before the final round due to these estimation errors.

MinResouces/DatasetSize	Factor	TotalSuccess	MeanSpeedup	StdSpeedup
1/8	2	6	1.62	0.26
1/4	2	10	1.11	0.12
1/9	3	2	2.51	0.46
1/3	3	10	1.31	0.08

Table 2: HRS Results for Random Forest Model

MinResouces/DatasetSize	Factor	TotalSuccess	MeanSpeedup	StdSpeedup
1/8	2	1	0.95	0.13
1/4	2	4	0.8	0.08
1/9	3	0	1.44	0.26
1/3	3	5	0.97	0.07

Table 3: HRS Results for XGBoost Model

It is important to note the tradeoff between speed and performance as follows. On increasing the "Factor" setting, the HRS will have lesser rounds and will terminate more quickly but it may perform worse. This is because we are selecting a lesser fraction of hyperparameters for the next round which may result in better hyperparameters being missed due to estimation errors. Similarly, on decreasing the "MinResources", this method runs faster but may perform worse as the estimation errors increase for lesser data points.

To understand the above tradeoff, we perform experiments by using different values for the "MinResources" and the "Factor" settings for the HRS method. We compare the results with the RS baseline (for the same set of candidate hyperparameters) on the following two metrics :

1. Success: True if HRS is able to find the same best hyperparameters as RS
2. Speedup: $\frac{Time(RS)}{Time(HRS)}$

HRS Settings: We use the following 4 combinations for the (MinResouces/DatasetSize, Factor) settings- (1/8,2), (1/4, 2), (1/9,3), (1/3, 3)
Dataset and models: We use the "compass" [A.1] classification task from the tabular data benchmarks and run the search methods on the train split (70%). Like the previous experiments on Bayesian hyperparameter tuning, we use Randomforest Classifier and XGBoost Classifier Models here.
Hyperparameter Search space: We use the hyperparameter search space from the paper Grinsztajn et al. [2022] and sample 144 candidates for each experiment. To make sure that for a given experiment, both HRS and RS start from the same candidates we set the same random state for both methods. We repeat each experiment 10 times by varying this random state to get results for a variety of different starting sets of hyperparameter candidates.

Table 2 and table 3 show the results for HRS for Random Forest and XGBoost respectively. The total success column shows the success across the 10 repeats of each experiment and the MeanSpeedup and StdSpeedup columns show the mean and standard deviation of the speedup achieved across the 10 repeats. We can observe in Table 2 that for the (MinResouces/DatasetSize, Factor) equal to (1/4,2) and (1/3,3) we achieve a 100% success rate while achieving a very little speedup of 1.11x and 1.31x respectively. On the other hand, for the (1/8,2) and (1/9,3) settings we achieve a lesser success rate of 60% and 20% and a higher speedup of 1.62x and 2.51x respectively. This clearly shows the tradeoff between speed and performance for HRS.
We see a similar tradeoff for the XGBoost experiments. It can also be observed that for XGBoost, HRS is not even faster than the RS baseline in 3 out of the 4 experimental settings. It is important to state that there is no guarantee that HRS will be faster than RS. This is because although HRS starts with lesser data points, it proceeds in multiple rounds. These later rounds are more computationally expensive as they use more and more of the dataset. Therefore the total runtime of HRS may or may not be lesser than RS.
Thus, for the settings where HRS gives good results, the speedup is not significant while for the settings it gives a good speedup, the results are poor.

4.2 Feature Encoding Methods

We experimented with the four different feature encoding techniques with one-hot encoding as our baseline for comparison and inference. We selected two datasets - nyc-taxi-green-dec-2016 and particulate-matter-ukair-2017 [A.1] - for the regression task and two datasets - KDDCup09-upselling and rl [A.1] – for the classification task. All four datasets chosen have several categorical columns, which are nominal in nature and show high cardinality (more than 20 levels). Additionally, we ensure that we remove all missing data from the datasets. In practice, we first remove columns containing many missing data, then all rows containing at least one missing entry.

As discussed above, the models chosen for the experimentation are Random Forest and XGBoost ensembles. We use the default hyperparameter setting for both models as the focus here is to independently assess and evaluate the impact of the encoding techniques on the models' performance. Therefore, for each dataset, we ran two ensemble models with five feature encoding techniques - One-Hot, Rare Label, Frequency, Hash, and Target - giving us a total of 40 different experimental settings (4 datasets × 2 models × 5 feature encoding techniques). The metric chosen for the classification tasks was accuracy, and for the regression task, R^2 score was chosen.

We compile the results of these experiments as in Figure 2. We can observe that one-hot encoding is clearly outperformed in some cases whereas it still remains the best encoding method in other scenarios. We also see that target encoding may not be the best choice when handling classification tasks. The performance of the models when features are encoded using the rare label encoding technique is highly comparable to and sometimes better than one-hot encoding. Varying the threshold based on the dataset can thus be beneficial. Moreover, models using one-hot encoding significantly takes the longest in terms of runtime to fit the data.

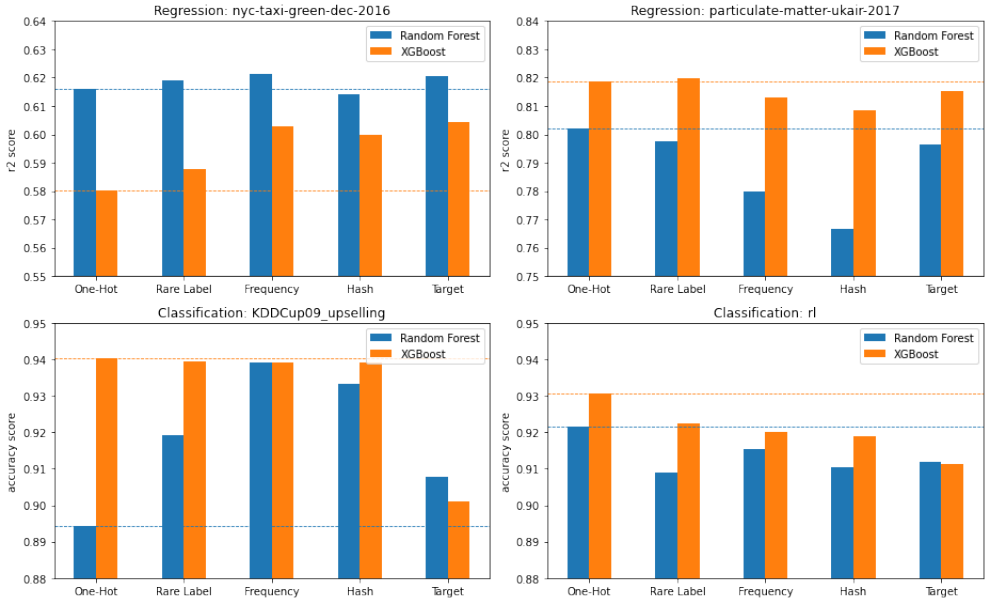

Figure 2: Comparison of different Feature Encoding Techniques with One-Hot Encoding over different datasets and ensemble models

5 Discussion and Conclusion

5.1 Limitations and Future Work

Firstly, for all Bayes Search methods, we have run the algorithms for just 40 iterations. This was done due to limited time and resources for this project. However, we can test this further for even better results for Bayes optimization approaches, as argued by Bergstra et al. [2013], when we increase the iteration count to thousands. Additionally, we can also increase the number of points being used simultaneously to construct the surrogate function in Bayes optimization to see the convergence

speed relation with it.
Secondly, for the Halving Random Search method, we were able to run the experiments for the compass dataset only and it would be important to repeat the results with more datasets to get more generalisable results.
Thirdly, while experimenting with the feature encoding methods, we have directly used default models to evaluate the performance. This can be extended to applying the hyperparameter search methods to the models and testing the various feature encoding methods on the best models obtained. We can also tweak the hyperparameters of the feature encoding techniques themselves, like the threshold for grouping of categories in rare label encoding.
Lastly, it can be useful to repeat these experiments for other and wider range of datasets to see if these conclusions are generalizing well for most of the scenarios.

5.2 Conclusion

In this project, we have provided an in-depth comparison of different hyperparameter optimization algorithms and feature encoding methods. We found that Tree Parzen Estimation based Bayesian Search outperforms Random Search and Gaussian process based Bayesian Search in terms of fewer iteration count needed to find the optimal solution. The final value of the metric was almost similar or better in TPE. GP based Bayes Search performs either similar to Random Search or slightly worse in some cases.
For Halving Random Search, we observe that for the settings where we get good results, the speedup is not significant while for the settings it gives a good speedup, but the results are poor.
In feature encoding methods, we observe that no one method of encoding gives the best results in all settings. The performance is dependent on the datasets, specifically the cardinality of the categorical columns. We thus infer that it is important to explore and experiment with various feature encoding methods and proceed with the one which gives the best performance for the use case.

5.3 Team Contribution

Authors are listed in alphabetical order. All members contributed equally throughout the duration of the project. All decisions related to the different parts of the project (like problem formulation, literature review, experimental design, and inferences) were taken collectively after discussion. Some specific contributions are as follows: Adeem took the initiative to implement the HRS method and run experiments. Apoorv did the same with GP based Bayes Search. Naveen was responsible for implementing and experimenting with TPE based Bayes Search. Anusha took charge of the evaluation of different feature encoding techniques.

References

Wikipedia: Gaussian process. URL `https://en.wikipedia.org/wiki/Gaussian_process`.

Benchmarking categorical encoders. URL `https://towardsdatascience.com/benchmarking-categorical-encoders-9c322bd77ee8`.

All about categorical variable encoding. URL `https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02`.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

Patricio Cerda and Gaël Varoquaux. Encoding high-cardinality string categorical variables, 2022.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, 2022. URL https://arxiv.org/abs/2207.08815.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pages 240–248. PMLR, 2016.

Will Koehrsen. A conceptual explanation of bayesian hyperparameter optimization for machine learning. URL https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f.

Florian Pargent, Florian Pfisterer, Janek Thomas, and Bernd Bischl. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, 37(5):2671–2692, mar 2022. doi: 10.1007/s00180-022-01207-6. URL https://doi.org/10.1007%2Fs00180-022-01207-6.

Cedric Seger. An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing, 2018.

Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications, 2020. URL https://arxiv.org/abs/2003.05689.

A Appendix

A.1 Datasets used

We describe in table 4 all the datasets used in our study, along with OpenML links to these dataset

A.2 Libraries used

- scikit-learn - `https://scikit-learn.org`
- HyperOpt-Sklearn - `https://github.com/hyperopt/hyperopt-sklearn`
- Scikit-Optimize - `https://github.com/scikit-optimize/scikit-optimize`
- Category Encoders - https://contrib.scikit-learn.org/category_encoders
- Feature Engine - `https://feature-engine.readthedocs.io/`

Dataset	**Instances**	**Features**	**Task**	**Link**
compass	16644	18	Classification	`https://www.openml.org/d/44162`
california	20640	8	Regression	`https://www.openml.org/d/44025`
house_sales	21613	17	Regression	`https://www.openml.org/d/44066`
nyc-taxi-green-dec-2016	581835	18	Regression	`https://openml.org/d/42729`
particulate-matter-ukair-2017	394299	9	Regression	`https://openml.org/d/42207`
KDDCup09_upselling	50000	230	Classification	`https://openml.org/d/1114`
rl	31406	22	Classification	`https://openml.org/d/41160`

Table 4: Datasets Used for Experimentation