STAT 4250: Applied Multivariate Analysis

Predicting Room Occupancy with Machine Learning Classifiers

Professor:
Dr. Yuan Ke

Author:
Bryce Davis

2 May 2021

Contents

1 Introduction **1**

2 EDA **1**

3 Random Forest Classification **3**

4 Gradient Boosting **5**

5 Discussion **7**

List of Figures

1 Target Distribution . . . 1
2 Correlation Heatmap . . . 2
3 Kernel Density Estimations . . . 2
4 Distribution of Light before(left) and after(right) a log-transform . . . 2
5 Sample, Single Decision Tree Classifier . . . 3
6 Confusion Matrix (left) and AUCROC (right) on default RF Classifier 4
7 Confusion Matrix (left) and AUCROC (right) on optimized RF Classifier . . . 5
8 Confusion Matrix (left) and AUCROC (right) on initial XGB Classifier 5
9 Confusion Matrix (left) and AUCROC (right) on optimized XGB Classifier . . . 6
10 FFNN Layer Design . . . 7
11 Confusion Matrices from FFNN Classifier . . . 8

List of Tables

1 GridSearchCV optimized hyper parameters for RF Classifier . . . 4
2 Optimal Hyper Parameters for XGBoost . . . 6

1 Introduction

Our Dataset comes from Candanedo and Feldheim[2], where they performed a study to accurately detect occupancy of an office building. Our variables are:

- Date: in the form: year-month-day hour:minute:second
- Temperature: in Celsius
- Relative Humidity: in %
- Light: in Lux
- CO2: in ppm
- Humidity Ratio: a derived quantity from temperature and relative humidity; in kg(water-vapor)/kg(air)
- Occupancy: 0 or 1; 0 for not occupied, 1 for occupied status. Ground-truth occupancy was obtained from time stamped pictures of the rooms that were taken every minute.

In this analysis, Occupancy is used as the target variable.

2 EDA

Initial observations show that this is a promising and well organized dataset. Figure 1 shows that our data has a slight class-imbalance, but we wouldn't classify "Occupied (1)" as a rare observation. We have a sufficient number of Occupied labels with 1729 (21.23% of total). The "Date" variable will cause us some trouble,

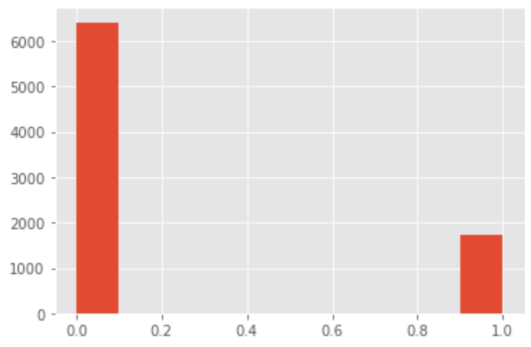


Figure 1: Target Distribution

however. Date has high cardinality with every single timestamp being a unique value. Initially I converted this to a Datetime object and transformed it into two separate dummy categories - hour and dayofweek. I though that since this is an office building, it would be more likely that the rooms would be occupied during the 9-5 hours and especially during the work-week. However, this resulted in a large amount of noise and each hour/day had no correlation with our target and thus it was removed entirely. The overall correlation matrix using Pearson's r can be found below in figure 2, notice how the hour is only correlated with itself. Unsurprisingly, humidity ratio is moderately correlated with humidity. However,

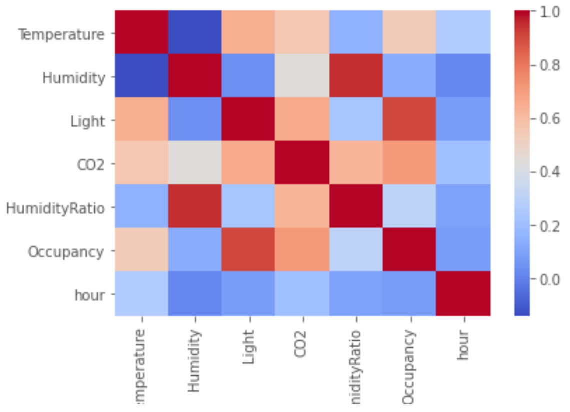


Figure 2: Correlation Heatmap

they do not exactly correlate equivalently with the other variables so they both will be kept. Taking a look at some Kernel Density Estimation plots can tell us about how our data varies given room occupation.

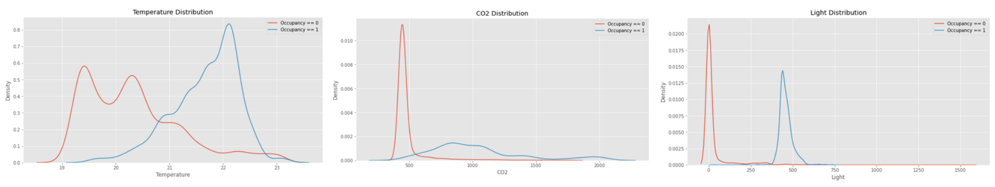


Figure 3: Kernel Density Estimations

Figure 3 above displays how influential our target is on the distribution of our variables. Temperature, Light, and CO2 are highly influenced by room occupancy. Though, the CO2 and Light distributions seem suitable for a log transform.

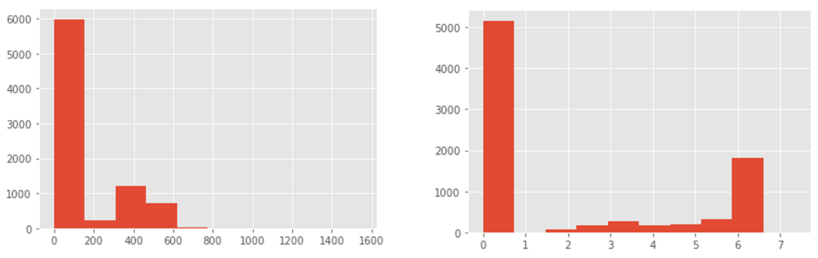


Figure 4: Distribution of Light before(left) and after(right) a log-transform

After this log transformation, all variables were then standard scaled to fit a N(0,1) distribution.

3 Random Forest Classification

Random Forests are an ensemble, tree-based learning method. A decision tree is fit on a randomized, bootstrapped sample of our data, hence the name "Random". The "Forest" part explains the ensemble nature of this classifier - where multiple trees will make decisions on each classification, and the majority vote of the "Forest" will be the classification of our observation. An Example of a single decision tree fit on our dataset is shown below. Depending on the values of our predictors (and to minimize error on the bootstrapped sample) the tree "branches" into "leaves". In order to make a classification, you just follow the path of the tree depending on the values of your observation. The node that the observation ends up on is colored to indicate which target it belongs to. [4] The performance metrics

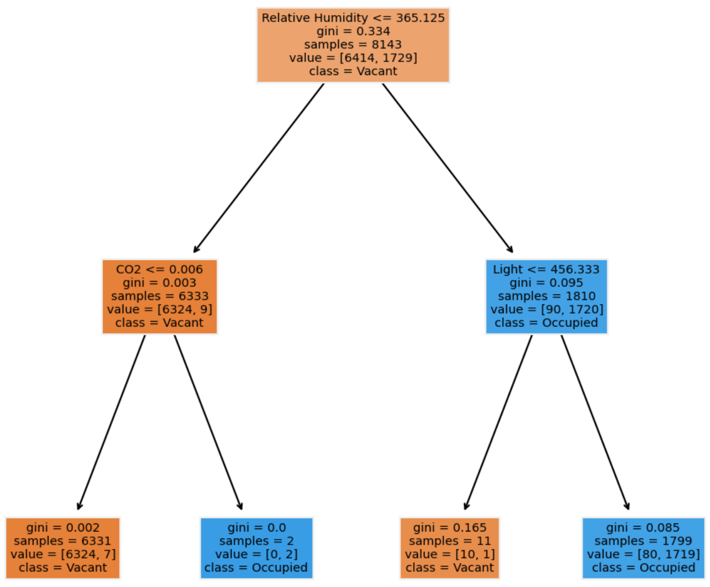


Figure 5: Sample, Single Decision Tree Classifier

we investigated were F1 Score, area under the receiver operating characteristic curve(AUCROC), precision, and recall. These are all metrics based on the number of true positives, true negatives, false positives, and false negatives. AUCROC can be interpreted as the chance of correctly prediciting a random observation.I decided to optimize for F1 Score because of its robustness for class imbalance. An initial fit with the SK-Learn RandomForestClassifier [6] was very promising. We begin with 99% accuracy, and an F-1 Score of 0.9713. The formula for F-1 is given below.

$$\text{F1} = 2 * \frac{Precision*Recall}{Precision+Recall} = 0.9713$$

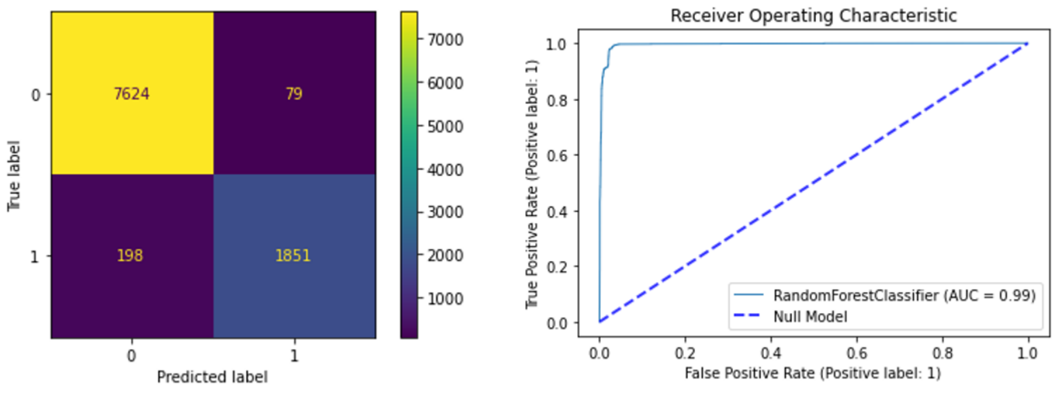


Figure 6: Confusion Matrix (left) and AUCROC (right) on default RF Classifier

Though, it is highly probable that we can expect better results with fine-tuning of our hyper-parameters. For random forests, the hyper-parameters define how our trees "grow". For example, the parameter "max depth" explains how many nodes deep our tree is allowed to be, "num estimators" represents the number of trees in our forest, etc. To optimize, we will use grid-search. For this method, you specify the ranges of each hyper-parameter and form a "grid". Then, using cross-validation, the model quickly iterates through all combinations of parameters on your grid and returns the selection that optimize your evaluation metric - F1 in our case. Table 1 Optimization provided a significant increase to model

Max Depth	Max Features	Min Samples Leaf	n_estimators	Min Sample Split
3	2	3	100	12

Table 1: GridSearchCV optimized hyper parameters for RF Classifier

performance. Our F-1 Score increased to 0.99, and our overall error was reduced from 2.8% to 0.68%. We only had 67 misclassifications out of nearly 10,000 observations! Figure 7 below can visualize this difference.

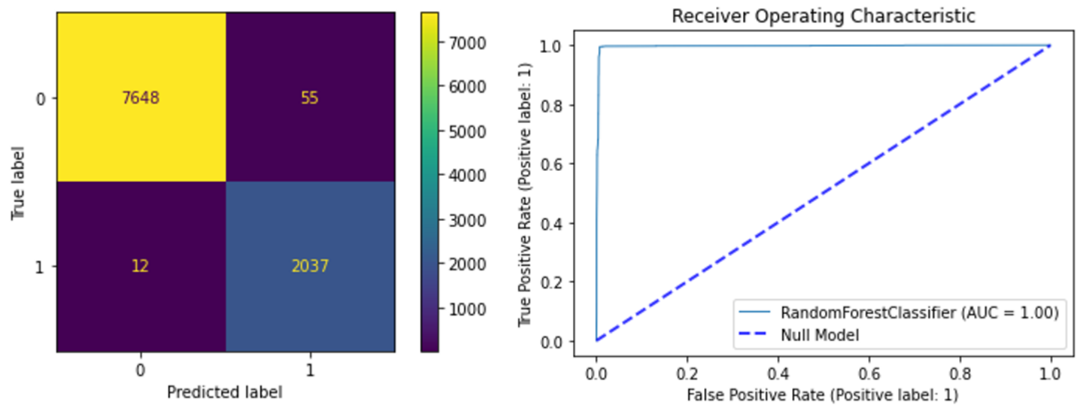


Figure 7: Confusion Matrix (left) and AUCROC (right) on optimized RF Classifier

4 Gradient Boosting

Gradient Boosting is another ensemble tree classifier, but in a different way. Graident Boosting is built upon on ensemble of ”weak” classifiers. The main idea is to fit an initial tree, and then use all subsequent trees to help classify examples the initial tree got wrong. Contrast this to Random Forests, which are called ”Strong” learners because each grew grows independently of one another. In this analysis we will be using XGBoost [3] (Extreme Gradient Boosting). XGBoost has an extensive amount of hyper parameter in comparison to random forests. This allows XGBoost to be used in a large variety of situations and thus has become a go-to of many data scientists and Kaggle competition connoisseurs. The initial fit using all default values was actually much worse than our RF initial fit - we receive an F-1 score of 0.89, and an overall accuracy of 0.95. Again, we can expect to get

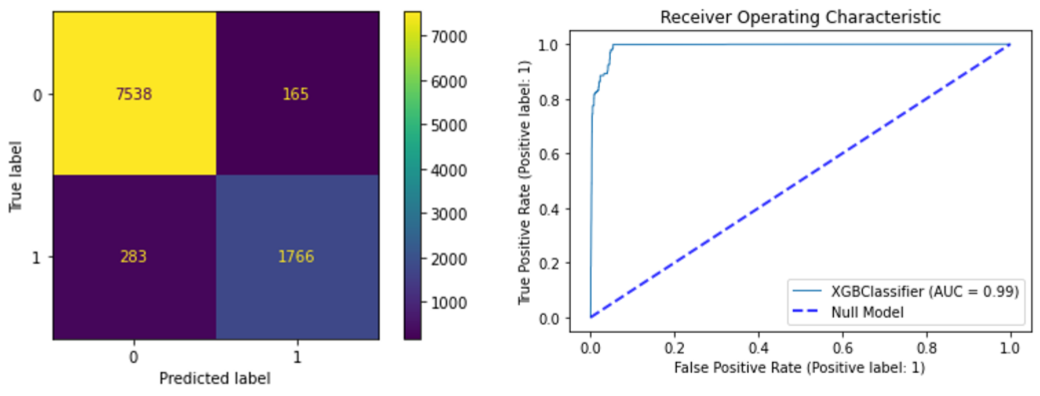


Figure 8: Confusion Matrix (left) and AUCROC (right) on initial XGB Classifier

more perfmorance from the XGBoost framework by optimizing for its large number of hyperparameters. In contrast to our earlier method, we will not be using GridSearch. GridSearch specifies a range of possible hyper parameters (a grid) and then finds the best subset of all combinations. While effective, it is computationally a brute force method and is extremely time consuming - especially with XGBoost's large number of possible hyper parameter situations. In contrast, Bayesian optimization, a.k.a Sequential Model-Based Optimization (SMBO), implements hyper parameter optimization by building a probability model of the objective function that maps the parameter input values to a probability of a loss. This surrogate probability model is easier to optimize than the actual objective function. The concept is to limit evaluation of the actual objective function by spending more time choosing the next values to try instead of actually trying them all. Bayesian Reasoning means updating a model based on new evidence, and, with each evaluation, the surrogate is re-calculated to incorporate the latest information[5]. Thus, the longer the algorithm runs, the closer the surrogate is to the actual objective function. This was computed using the BayesianOptimization package. In my own testing, GridSearchCV would take 9 hours to fully search the same parameter space that the Bayesian Optimizer searched in just 20 minutes. The Bayesian Optimized hyper parameters are found in table 2.

Variable Name	Value
base_score	0.2902
gamma	1.3395
learning_rate	0.3698
max_depth	35.5524
min_child_weight	1.1177
reg_alpha	0.0408
reg_lambda	0.0183
scale_pos_weight	38.8599

Table 2: Optimal Hyper Parameters for XGBoost

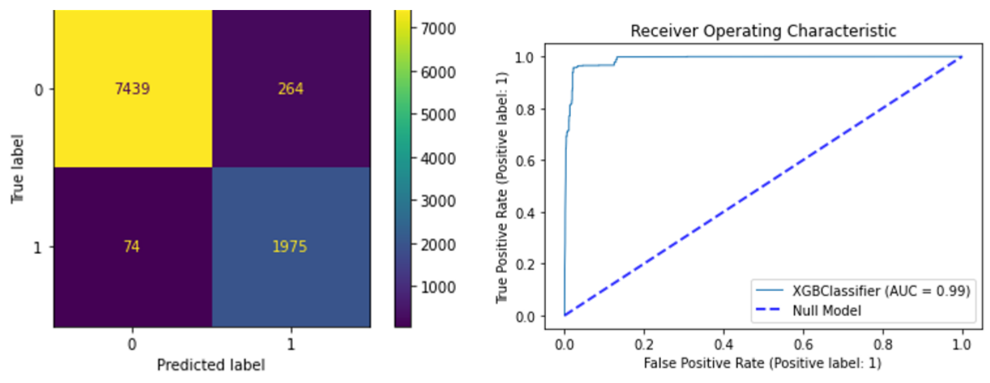
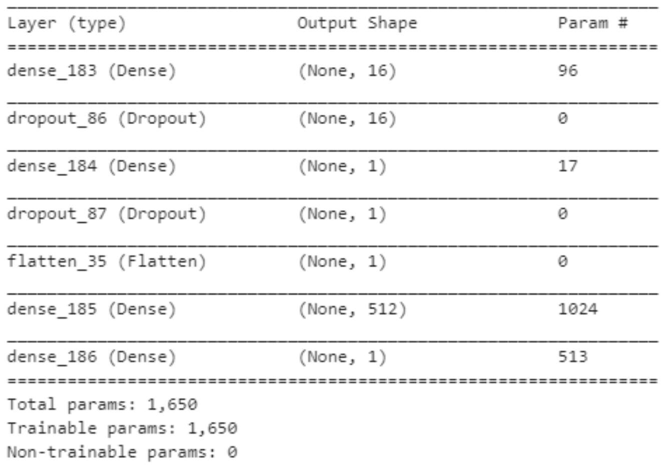


Figure 9: Confusion Matrix (left) and AUCROC (right) on optimized XGB Classifier

Figure 9 was surprising, especially given how well the random forest performed. Although optimization increased our F-1 score to 0.9659 and our accuracy to 0.97, this classifier is not performing nearly as well as our random forest.

5 Discussion

Between random forests and gradient boosting, our optimized random forest model was clearly superior at predicting room occupancy. Though, this is probably due to human error on my behalf. Random Forests are pretty simple in the grand-scheme of machine learning and only have 5 possible hyper parameters to tune. In contrast, XGBoost has nearly 20 possible, tunable parameters. GridSearchCV would have definitely out performed Bayesian Optimization but my computer would always crash before the computation could complete! Therefore, it was a necessary choice to use Bayesian. I think this really displays a highlight of this optimization strategy. I was not able to achieve maximum performance (due to computation limitations), but I was able to gain a large performance boost with far less time spent on the actual optimization step. Additionally, more complex does not equate to better performance. Our dataset is pretty well behaved, and room occupation was highly correlated with individual variables so maybe using machine learning is completely overkill for this application. I learned this the hard way when trying to fit a Feed-Forward Neural Network (FFNN) to this problem. I was interested in using Tensorflow [1] as a binary classifier, and my initial model set up is shown below. I thought this would be my ticket to a perfect classification,



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_183 (Dense)            (None, 16)                96
_________________________________________________________________
dropout_86 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_184 (Dense)            (None, 1)                 17
_________________________________________________________________
dropout_87 (Dropout)         (None, 1)                 0
_________________________________________________________________
flatten_35 (Flatten)         (None, 1)                 0
_________________________________________________________________
dense_185 (Dense)            (None, 512)               1024
_________________________________________________________________
dense_186 (Dense)            (None, 1)                 513
=================================================================
Total params: 1,650
Trainable params: 1,650
Non-trainable params: 0
```

Figure 10: FFNN Layer Design

especially given the performance of the Random Forests. I could not have been more wrong. Epoch after epoch the FFNN would be stuck in local minima, only classify ”Vacant” all 10000 times, etc. Not only was I experiencing issues such as over fitting, but also I could not understand why the model was making its incorrect classifications. Attempting to alleviate this problem only made things even more confusing. I added two dropout layers, where a percentage of the layer's neurons die randomly so that information can be better spread across the whole model. I added regularization, which helps with keeping the neuron weights from being stuck at 0. I also tweaked the model complexity in order to simplify the structure as well. As you can see from the confusion matrices, nothing I did seemed to make this model competitive with the tree-based classifiers. Using

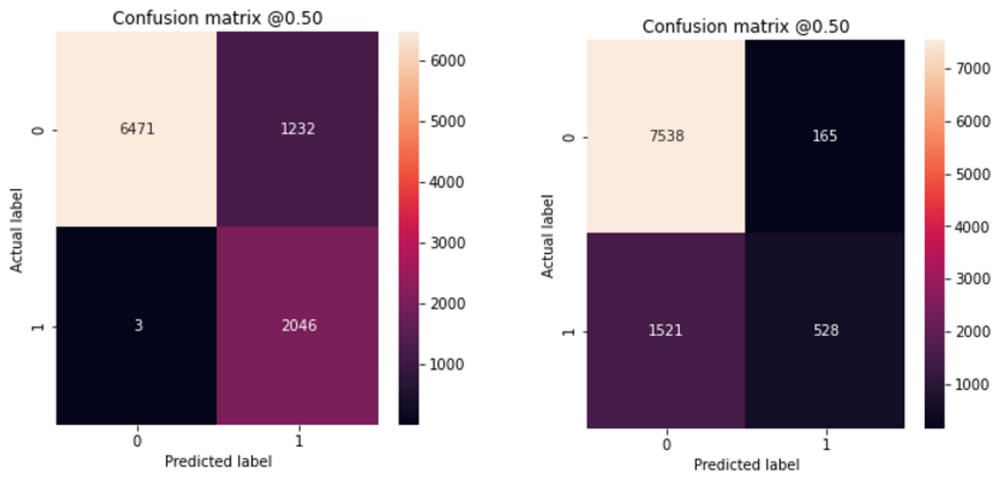


Figure 11: Confusion Matrices from FFNN Classifier

FFNNs on this dataset is certainly an example of machine learning overkill. This is a pretty simple problem and we have displayed that there are pretty obvious signs of room occupancy. Neural Networks are incredibly powerful, but they are particularly useful when there is some form of hierarchical structure within the data, such as text or images. Although there is definitely a better approach to applying FFNN to this problem, the amount of time required to fine tune a highly complex model is not worth it. Even though we examined three separate applications of machine learning, their performances only decreased as model complexity grew. Since our target class was not rare, random forests were able to capture the relationships between our variables just fine - and I would assume simple logistic regression would too.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Luis M. Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and Buildings*, 112:28–39, 2016.

[3] Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016.

[4] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1):2522–5839, 2020.

[5] Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *J. Mach. Learn. Res.*, 15(1):3735–3739, January 2014.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.