# Exoplanets detection using
# auto-tuned random forest

Lorenzo Loconte

**Abstract**

This work consists of identifying exoplanets using random forest which hyperparameters are automatically optimized with techniques that come from the `AutoML` research. The model is trained using the processed data that comes from the `NASA Kepler` project of discovering exoplanets (i.e. planets outside our solar system). The hyperparameters of the model are optimized and cross-validated with `Hyperband`, a simple yet effective and scalable method for hyperparameters optimization. The dataset used in this work can be found at **https://exoplanetarchive.ipac.caltech.edu**.

## 1  Introduction

**Kepler Object of Interest**  The dataset used in this work is the cumulative list of the `Kepler Object of Interest` (KOI) that comes from the `NASA Kepler` project of discovering exoplanets. The dataset is composed by a list of examples regarding exoplanets. Each example have a label indicating if the corresponding exoplanet is candidate, false positive or confirmed. The provided dataset consists of a lot of heterogeneous features. For the classification task the selected features can be found in Appendix B. The candidate exoplanets (i.e. the exoplanets which existence is uncertain) are discarded in order to reduce noise and improve the learning process. For simplicity all the examples having null values are removed from the dataset. Furthermore, all the selected features are numeric and so the preprocessing method applied to the dataset is the standard normalization. The resulting dataset contains `6884` samples, which ˜`67%` are false positives and the remaining are confirmed.

**Random Forests**  The model used for the classification task is a random forest, an ensemble of random decision trees. The prediction of the random forest is computed as the mode of each tree's classification. Random decision forests correct the habit of decision trees of overfitting. Each tree of the forest is trained on a subset of the features and samples. The hyperparameters of the random forest are optimized and cross-validated using approaches

that come from the `AutoML` research. The following hyperparameters are optimized:

- The split criterion (Gini impurity or information gain, see Appendix A for details)

- The percentage of features to use for each tree

- The maximum depth of each tree

- The minimum number of samples required to split an internal node

- The minimum number of samples to be at a leaf node

As the next section will expose, the number of trees in the random forest is not considered an hyperparameter because it represents the budget of the hyperparameters optimization algorithm chosen. In this work we refer with budget as the computational cost of cross-validating a certain model.

## 2    Hyperparameters Tuning

In this section it will be shown a formalization of the hyperparameters optimization task as a search problem. In this work we refer to hyperparameters as the ones that cannot be trained (e.g. the ones described in the previous section). After that we briefly introduce the hyperparameters optimizer used in this work, `Hyperband`, as described in [1].

Given a training set $T$, a model $L$ and its hyperparameters search space $\Omega$, the hyperparameters optimization task consists to find a set of hyperparameters $\omega \in \Omega$ such that the resulting model $L\langle\omega\rangle$ maximizes a certain score $S$ using cross-validation on $T$.

There are a lot of different techniques for hyperaparameters optimization, some of them are the following:

- Random Search

- Grid Search

- Hyperband

- Bayesian Optimization

- Hybrid Approaches (like `BOHB` [2])

The simplest algorithm is `Random Search`. It works well if the hyperparameters search space is not too big. `Random Search` is expensive because each evaluation of the score function is done on models completely built from scratch. So, each evaluation of the score function on some randomly selected hyperparameters will use the same amount of resources, even if

some hyperparameters of the search space obtain a very low score. The idea behind `Hyperband` is that it's better to save up resources on these hyperparamters configurations that doesn't obtain a sufficiently good score by gradually building more complex models over time.

As we said before we refer to budget as the computational cost of cross-validating a certain model. For example, for neural networks the budget can be the number of training epochs, while for random forests it can be the number of trees.

Given a budget $R$, `Hyperband` consists of random sampling some points in the hyperparameters space and cross-validating them using only a fraction of the budget $R$. After that we pick a fraction $1/\eta$ of the best models. In this way we discard the models that don't obtain a sufficiently good score saving up computational resources. After that we strengthen the previous models using a larger portion of $R$ and iterate in this way until we use the complete budget $R$. The procedure of consequently eliminating hyperparameters samples that don't obtain a good score is called `Successive Halving`. The fraction of the budget to use for each iteration grows geometrically in respect of $\eta$. The problem is to determinate the number of initial random points of the hyperparameters space to sample. To resolve this issue we execute the previous discussed algorithm multiple times using less and less initial random points and using an higher starting budget.

Even if it is a simple algorithm, it is well scalable on multiple CPUs because we assume that every point in the hyperparameters space is independent from the others. The following algorithm shows an implementation of $Hyperband$.

---

**Algorithm 1** Hyperband algorithm for hyperparameters optimization

---

1: **procedure** $\textsc{Hyperband}(T, L, S, \Omega, R, \eta)$
2: $\quad C \leftarrow \emptyset$
3: $\quad s_{max} \leftarrow \lfloor \log_\eta R \rfloor, \quad B \leftarrow (s_{max} + 1)R$
4: $\quad$ **for** $s \in \{s_{max}, s_{max} - 1, ..., 0\}$ **do**
5: $\quad\quad n \leftarrow \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$
6: $\quad\quad r \leftarrow R\eta^{-s}$
7: $\quad\quad H \leftarrow getRandomHyperparameters(n, \Omega)$
8: $\quad\quad$ **for** $i \in \{0, ..., s\}$ **do**
9: $\quad\quad\quad n_i \leftarrow \lfloor n\eta^{-i} \rfloor$
10: $\quad\quad\quad r_i \leftarrow r\eta^i$
11: $\quad\quad\quad H \leftarrow top_{\lfloor n_i/\eta \rfloor} \arg sort_\omega \{S(L\langle \omega \cup \{r_i\}\rangle, T) \mid \omega \in H\}$
12: $\quad\quad$ **end for**
13: $\quad\quad C \leftarrow C \cup H$
14: $\quad$ **end for**
15: $\quad$ **return** $\arg max_\omega \{S(L\langle \omega \cup \{R\}\rangle, T) \mid \omega \in C\}$
16: **end procedure**

---

| Hyperparameters | Values |
|---|---|
| Split Criterions | $\{gini, entropy\}$ |
| Features fraction | $(0, 1]$ |
| Maximum Depth | $\{8, 9, ..., 32\}$ |
| Minimum samples to split | $\{2, 3, ..., 16\}$ |
| Minimum samples at a leaf | $\{1, 2, ..., 5\}$ |

Table 1: The hyperparameters space subset used in this work.

In our case the model $L$ is a random forest. As said before, the budget is the maximum number of trees that the final random forest will have. The choice of the score $S$ is dependent from the target task. For simplicity we use the $F_1$ score (i.e. we want to maximize both *Precision* and *Recall*). In this work the hyperparameters search space can be defined formally as:

$$\Omega = \overbrace{\{gini, entropy\}}^{\text{Split Criterion}} \times \overbrace{(0, 1]}^{\text{Features fraction}} \times$$

$$\times \overbrace{\mathbb{N}^*}^{\text{Max depth}} \times \overbrace{\mathbb{N}^*}^{\text{Min samples to split}} \times \overbrace{\mathbb{N}^*}^{\text{Min samples at leaf}}$$

For obvious reasons only a subset of this space can be explored. Table 1 shows the subset of the search space used in this work.

The result of the hyperparameters optimization task is a point in the hyperparameters space:

$$\omega = (c, p, h, k, t) \in \Omega$$

where $c$ is the split criterion, $p$ is the features fraction for each tree, $h$ is the maximum depth, $k$ is the minimum number of samples to split and $t$ is the minimum number of samples at a leaf.

# 3   Conclusion

As you can see from Table 2 the auto-tuned random forest used in this work obtained a way better *Precision* and *Recall*. We can choose any score function for hyperparameters optimization. For example we can decide to maximize only one between *Precision* and *Recall* scores. Furthermore, it's possible to use other metrics like *Average Precision*.

As you can see from Table 3 the false positives count (the number of examples, which are predicted as exoplanets, but that are not) and the false negatives count (the number of actual exoplanets not being discovered), are pretty low.

In the end we present Table 4 that shows the importances of the first five out eighteen features used.

| Model | Precision | Recall | $F_1$ |
|---|---|---|---|
| k-NN | 0.700 | 0.832 | 0.757 |
| SVC | 0.775 | 0.854 | 0.808 |
| 2-layer NN | 0.830 | 0.852 | 0.838 |
| **Random Forest w/Hyperband** | **0.866** | **0.874** | **0.870** |

Table 2: *Precision, Recall* and $F_1$ metrics of several models for comparison. The models hyperparameters are the default ones implemented in `sklearn`. For `Hyperband` we used $R = 216$ and $\eta = 3$.

|  | **Actual** | |
|---|---|---|
|  | Positive | Negative |
| **Predicted** Positive | **402** | 62 |
| **Predicted** Negative | 58 | **855** |

Table 3: Confusion matrix over the test set.

| Feature # | Description | Importance |
|---|---|---|
| 5 | Planetary Radius | 0.276 |
| 1 | Orbital Period | 0.162 |
| 4 | Fitted Stellar Density | 0.087 |
| 3 | Transit Depth | 0.085 |
| 6 | Inclination | 0.071 |

Table 4: Features importances in descending order.

# Appendices

## A    Decision tree split criterions

### A.1    Gini impurity

Citing `wikipedia.org`, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. To compute Gini impurity for a set of items with $J$ classes, suppose $i \in \{1, 2, ..., J\}$, and let $p_i$ be the faction of items labeled with class $i$ in the set.

$$I_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = \sum_{i=1}^{J} p_i(1 - p_i) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} {p_i}^2 = 1 - \sum_{i=1}^{J} {p_i}^2$$

### A.2    Information gain

Information gain is based on the concept of entropy and information content from information theory. For each node of the tree, the information value represents the expected amount of information that would be needed to specify whether a new instance should be classified yes or no, given the example reached that node. Given $T$ the set of training examples, suppose $i \in \{1, 2, ..., J\}$, where $J$ is the number of classes, and let $p_i$ be the faction of items labeled with class $i$ in the training set.

$$IG(T, a) = \mathrm{H}(T) - \mathrm{H}(T|a) = -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{a} p(a) \sum_{i=1}^{J} - \Pr(i|a) \log_2 \Pr(i|a)$$

## B    Exoplanet Features

1. Orbital Period [$days$]

2. Transit Duration [$hrs$]

3. Transit Depth [$ppm$]

4. Fitted Stellar Density [$g/cm^3$]

5. Planetary Radius [$Earth\,radii$]

6. Inclination [$deg$]

7. Equilibrium Temperature [$K$]

8. Insolation Flux [$Earth\,flux$]

9. Stellar Effective Temperature [$K$]

10. Stellar Surface Gravity [$\log_{10}(cm/s^2)$]

11. Stellar Radius [$Solar\,radii$]

12. Stellar Mass [$Solar\,mass$]

13. RA [$decimal\,degrees$]

14. Dec [$decimal\,degrees$]

15. Kepler-band [$mag$]

# References

[1] L. Li, K. Jamieson, Giulia DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18, 2018.

[2] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *ArXiv*, abs/1807.01774, 2018.