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`) tables that come 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 A. 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 the preprocessing method applied to the dataset is the standard normalization. The resulting dataset contains `6399` samples, which approximately `~66%` are false positives and the remaining are confirmed.

Random Forests The model used for the classification task is a random forest, a bagging ensemble of decision trees. The prediction of the random forest is computed as the mode of the predictions made by the decision trees. Each tree of the forest is trained on a subset of the features. Random forests correct the habit of decision trees of overfitting on their training set. In fact decision trees tend to overfit their training sets because they have a

low bias but an high variance. So, random forests generally obtain a better performance because they reduce the variance, with a little increase of bias due to the `Bias-Variance` tradeoff. 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 B for details)
- The fraction 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 Ω, the hyperparameters optimization task consists to find an hyperparameters configuration $\omega \in \Omega$ 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])

Of the techniques cited above only `Bayesian Optimization` guarantees the convergence to the global maximum in the hyperparameters search space (see [3] for details). The simplest algorithm is `Random Search` and 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 computational 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.

Algorithm 1 Hyperband algorithm for hyperparameters optimization

procedure HYPERBAND(T, L, S, Ω, R, η)
  $C \leftarrow \emptyset$
  $s_{max} \leftarrow \lfloor \log_\eta R \rfloor, \quad B \leftarrow (s_{max}+1)R$
  for $s \in \{s_{max}, s_{max}-1, ..., 0\}$ **do**
    $n \leftarrow \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil, \quad r \leftarrow R\eta^{-s}$
    $H \leftarrow getRandomHyperparameters(n, \Omega)$
    for $i \in \{0, ..., s\}$ **do**
      $n_i \leftarrow \lfloor n\eta^{-i} \rfloor, \quad r_i \leftarrow r\eta^i$
      $H \leftarrow top_{\lfloor n_i/\eta \rfloor} \arg sort_\omega \{S(L\langle \omega \cup \{r_i\} \rangle, T) \mid \omega \in H\}$
    end for
    $C \leftarrow C \cup H$
  end for
  return $\arg max_\omega \{S(L\langle \omega \cup \{R\} \rangle, T) \mid \omega \in C\}$
end procedure

The algorithm above shows an implementation of `Hyperband` as desribed now. Given a budget R, `Hyperband` consists of random sampling some points (using an uniform distribution) 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 didn'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 η.

The remaining problem is to determinate the number of initial random points of the hyperparameters space to sample and the initial budget to use. Generally the number of initial random points is $\eta^{\lfloor \log_\eta R \rfloor}$ and the initial budget is set to $R\eta^{-\lfloor \log_\eta R \rfloor}$. Furthermore, we execute the discussed algorithm multiple times using less and less initial random points and using an higher starting budget. So, initially *exploration* is preferred and *exploitation* is performed later.

Even if `Hyperband` 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.

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 the F_1 score is used (i.e. we want to maximize both *Precision* and *Recall*). In this work the hyperparameters search space can be defined formally as:

$$\Omega = \overbrace{\{gini, entropy\}}^{\text{Split Criterion}} \times \overbrace{(0, 1]}^{\text{Features fraction}} \times \overbrace{\mathbb{N}^*}^{\text{Max depth}} \times \overbrace{\mathbb{N}^*}^{\text{Min samples to split}} \times \overbrace{\mathbb{N}^*}^{\text{Min samples at leaf}}$$

For obvious reasons only a subset of this space can be explored. Table 1 shows the subset of the hyperparameters 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, such that the random forest built $L\langle\omega\rangle$ maximize the cross-validation score S on the training set T.

Hyperparameters	Values
Split Criterions	$\{gini, entropy\}$
Features fraction	$(0, 1]$
Maximum Depth	$\{8, 9, ..., 32\}$
Minimum samples to split	$\{2, 3, ..., 8\}$
Minimum samples at a leaf	$\{1, 2, ..., 4\}$

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

Model	Precision	Recall	F_1
k-NN	0.696	0.816	0.747
SVC	0.791	0.845	0.813
2-layer NN	0.840	0.863	0.849
Random Forest	0.871	0.843	0.850
Random Forest w/HB	**0.882**	**0.900**	**0.891**

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` $R = 216$ and $\eta = 3$ are used.

		Actual	
		Positive	Negative
Predicted	Positive	**397**	53
	Negative	44	**786**

Table 3: Confusion matrix over the test set.

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*. 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. At contrary of other models, like neural networks, that are *black-box* models, random forests are *white-box* models: the predictions can be explained deeply and it is possible to evaluate the importance of each feature used for predictions. In the end we present Table 4 that shows the importances of the first five most important features.

Feature #	Description	Importance
5	Planetary Radius	0.226
3	Transit Depth	0.105
1	Orbital Period	0.095
6	Inclination	0.082
4	Fitted Stellar Density	0.071

Table 4: Features importances in descending order.

Appendices

A Exoplanet Selected 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]
16. g-band [mag]
17. r-band [mag]
18. i-band [mag]
19. z-band [mag]
20. J-band [mag]
21. H-band [mag]
22. K-band [mag]

B Decision tree split criterions

B.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.

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

B.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.

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

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.

[3] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.