Monthly Notices of the ROYAL ASTRONOMICAL SOCIETY

MNRAS **513,** 5505–5516 (2022) https://doi.org/10.1093/mnras/stab3692
Advance Access publication 2021 December 21

Exoplanet detection using machine learning

Abhishek Malik,★ Benjamin P. Moster ★ and Christian Obermeier
Universitäts-Sternwarte, Ludwig-Maximilians-Universität München, Scheinerstr 1, D-81679 München, Germany

Accepted 2021 October 13. Received 2021 October 12; in original form 2021 March 4

ABSTRACT

We introduce a new machine learning based technique to detect exoplanets using the transit method. Machine learning and deep learning techniques have proven to be broadly applicable in various scientific research areas. We aim to exploit some of these methods to improve the conventional algorithm based approaches presently used in astrophysics to detect exoplanets. Using the time series analysis library *TSFRESH* to analyse light curves, we extracted 789 features from each curve, which capture the information about the characteristics of a light curve. We then used these features to train a gradient boosting classifier using the machine learning tool *LIGHTGBM*. This approach was tested on K2 campaign 7 data with injected artificial transit signals, which showed that it is competitive compared to the conventional box least-squares fitting method. We further found that our method produced comparable results to existing state-of-the-art deep learning models, while being much more computationally efficient and without needing folded and secondary views of the light curves. For Kepler data, the method is able to predict a planet with an AUC of 0.948, so that 94.8 per cent of the true planet signals are ranked higher than non-planet signals. The resulting recall is 0.96, so that 96 per cent of real planets are classified as planets. For the Transiting Exoplanet Survey Satellite data, we found our method can classify light curves with an accuracy of 0.98, and is able to identify planets with a recall of 0.82 at a precision of 0.63.

Key words: methods: data analysis – techniques: photometric – planets and satellites: detection.

1 INTRODUCTION

Since the discovery of the first exoplanets (Wolszczan & Frail 1992; Mayor & Queloz 1995), the field of planet detection has become a major research area in astrophysics. To date, over 4000 exoplanets have been discovered with various techniques, such as the radial velocity method (Campbell, Walker & Yang 1988; Latham et al. 1989), astrometry (Benedict et al. 2002; Muterspaugh et al. 2010), direct imaging (Chauvin et al. 2004; Thalmann et al. 2009), and gravitational microlensing (Mao & Paczynski 1991; Beaulieu et al. 2006). Yet, the majority of confirmed exoplanets have been discovered with the transit method (Charbonneau et al. 2000; Naef et al. 2001).

Depending on the position of the observer, a planet may move in front of its host star blocking a part of the star's light, which causes the observed visual brightness of the star to drop by a small amount, depending on the relative sizes of the star and the planet. For the transit method, the brightness of a star is thus continually observed, whereupon 'dips' in the obtained light curve are sought. One drawback of this method is that transits can only be observed when the planet's orbit is perfectly aligned from the location of the observer. Another issue is the high rate of false detections, e.g. through eclipsing binary systems or transits by planet sized stars, so that discoveries need to be confirmed with alternative methods.

NASA's Kepler mission (Borucki et al. 2010), and its second survey programme K2 (Howell et al. 2014), were a major step forward in the creation of a vast catalogue of exoplanet systems, which may give insights into the formation process of planets and the abundance of potentially habitable Earth-like analogues. Although Kepler was designed as a statistical mission to investigate the frequency of Earth-size exoplanets in or near habitable zones, most early results focused on individual systems (e.g. Batalha et al. 2011; Muirhead et al. 2012), resulting in rather homogeneous catalogues. Later studies shifted their focus towards the statistics of the exoplanet population (e.g. Fressin et al. 2013; Burke et al. 2015). However, the validation process of the candidates discovered in the Kepler mission is still an ongoing process, as ruling out false-positive detections is time-consuming and new candidates are still being discovered.

In 2018 April, NASA launched the Transiting Exoplanet Survey Satellite (TESS; Ricker et al. 2015) as a successor of Kepler with the primary objective to survey some of the brightest stars. It covers 85 per cent of the sky – an area that is 400 times larger than that covered by Kepler – and produces around one million light curves per month. TESS observes stars that are 30–100 times brighter than those selected by the Kepler mission. Consequently, it is possible to identify targets that are far easier to follow up for detailed observation with other space-based and ground-based telescopes. Some of the early results of TESS include the discovery of a super-Earth in the Pi Mensae system (Huang et al. 2018), an ultrashort period planet orbiting a red dwarf (Vanderspek et al. 2019), and an Earth-sized exoplanet in the habitable zone (Gilbert et al. 2020).

The first step in analysing observed light curves is to search for periodic signals that may be consistent with transiting planets, so-called threshold crossing events (TCEs). These TCEs are then inspected to eliminate erroneously selected non-signals (e.g. instrumental noise or astrophysical variability), either manually by humans or in an automated way (e.g. Coughlin et al. 2016a). A typical phase-folded

★ E-mail: moster@usm.lmu.de (BPM); a.malik@usm.lmu.de (AM)

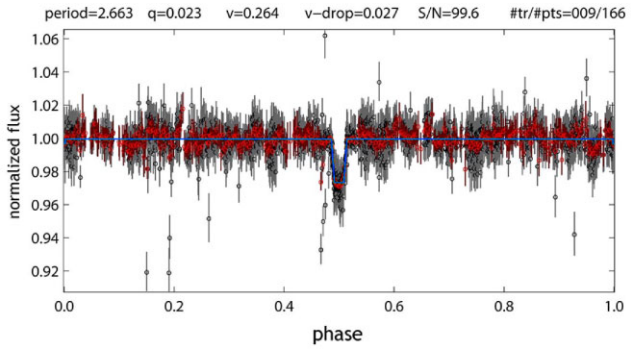

Figure 1. Phase-folded light curve of a K dwarf being orbited by a hot Jupiter candidate, observed by the Pan-Planets survey. The BLS method has been used to fit the light curve. Shown at the top are the resulting parameters: period (days), transit duration q (in units of phase), transit v shape (0 corresponds to a box, 1 to a V), transit light drop, signal-to-noise (S/N), and number of transits/number of points in the transits. Binned data points are shown in red, while the green line shows the best-fitting model. See Obermeier et al. (2016) for more details.

light curve from the Pan-Planets survey (Obermeier et al. 2016) is shown in Fig. 1.

One of the most commonly used methods to detect exoplanet in light curves is box-fitting least squares (BLS; Kovács, Zucker & Mazeh 2002), as it is very effective in detecting periodical signals that can be approximated by a two level system such as transits. In practice, a box model is fit to the light curve, which yields parameters such as the period, the transit duration, and the transit light drop. The green line in Fig. 1 shows the BLS fit to the light curve (Obermeier 2016). Cases with a seemingly good fit can then be manually reviewed. However, BLS is limited in terms of signal-to-noise and data cadence, and is vulnerable to false-positive detections created by cosmic rays, random noise patterns, or stellar variability.

Kepler, TESS, and other similar surveys still rely on manual analysis (see Yu et al. 2019) for a good overview of this process). For a typical TESS sector, usually a group of experts manually eliminate obvious false positive cases, a process that alone can take a few days. From the remaining cases, each one has to be viewed by at least three experts. This kind of procedure can lead to disagreements on a particular case, as even experts might not maintain the same definition for the classification. A difference of opinion may arise in a team of experts on a particular case due to external factors like the way a case is presented, other TCEs viewed recently, or something as little as the time of day or their mood (c.f. Coughlin et al. 2016a).

For these reasons, systems are needed that can reliably and repeatably select the most important planet candidates for us, which can then be manually reviewed for confirmation at a later stage. Over the past years, there was a growing interest in building automatic vetting systems with machine learning methods, as they provide computers the ability to learn from data without being explicitly programmed, and are instead able to detect patterns automatically. Some of the notable earliest attempts include the Robovetter (Coughlin et al. 2016b), the Autovetter project (McCauliff et al. 2015), and SIDRA (Mislis et al. 2016), where tree-based models trained for vetting. However, the research moved away from classical machine learning methods recently, and deep learning methods became the new focus. Among the most notable work in this area was done by Shallue & Vanderburg (2018). They introduced a novel deep learning architecture, *Astronet*, which produced very accurate results for the Kepler data. Their approach and their model architecture was adapted and applied to data from several different surveys such as Kepler's K2 mission (Dattilo et al. 2019), next-generation transit survey (NGTS; Chaushev et al. 2019), and TESS data (Yu et al. 2019).

Since the introduction of *Astronet* by Shallue & Vanderburg (2018), the community has moved on to deep learning, as deep learning methods tend to produce better results than classical machine learning approaches, especially for more complex problems or data types. However, deep learning models are usually computationally expensive and require large amounts of data to train properly. Often they are prone to overfitting the data if not regularized sufficiently. Moreover, it is usually difficult to understand, which input data (features) have been important to derive the results, which limits the physical interpretability. Therefore, in some cases, these models can be superfluous for the given problem and simpler or less computationally expensive approaches can perform equally well or even better, and can allow for a better understanding of the underlying problem. On the other hand, it is also important to sometimes move away from the currently best-performing methods and explore new alternatives. With this motivation, in this paper we propose a new direction to approach the problem of finding exoplanets in light-curve data using classical machine learning.

This paper is organized in five sections. Section 2 contains details about our methodology, specifically data preparation, feature extraction, and model training. Section 3 explains the results achieved on simulated, Kepler and TESS data. We also compare our results with some of the currently best-performing models. In Section 4, we compare the main differences how our model differs from the other deep learning based methods being used in the area. We also provide details of our vetting tool that can be used to make real time inferences. Finally, we conclude and discuss future steps in Section 5.

2 METHODS

Machine learning methods are widely used in scientific research areas to build classifiers, i.e. an algorithm that separates data into two or multiple classes. In our case, we are building a binary classifier that will separate each time series photometry, a so-called light curve, into the classes 'planet candidate' and 'non-candidate'. The state-of-the-art machine learning method for planet detection presented by Shallue & Vanderburg (2018) utilizes deep learning, a class of machine learning methods based on artificial neural networks. Our method on the other hand is based on classical machine learning. One essential difference between our approach and deep learning is that the latter is able to extract features automatically, while for classical machine learning the features have to be calculated beforehand and provided as input to the model. To this end, we use time series feature extraction based on scalable hypothesis tests (*TSFRESH*; Christ et al. 2018), a PYTHON-based library for feature extraction. The idea behind the method was inspired by methodology used in time series prediction projects employing feature engineering libraries. *TSFRESH* is also used in projects like machine fault prediction (LaCasse, Otieno & Maturana 2019), diagnosing Alzheimer's disease (Moore, Lyons & Gallacher 2019), identifying epileptic seizures in electroencephalography signals (Wu, Zhou & Li 2020), earthquake prediction (Khan, Choi & Kwon 2020), and time series forecasting for business applications (Abrishami 2019). Light curves are essentially a time series and so these tools can directly be used for our case.

We trained and tested our model on three kinds of data sets. The first stage used simulated data with K2 photometry as a baseline with additional injected transits. We then trained it on Kepler and finally TESS photometry. Each stage is split into three parts:

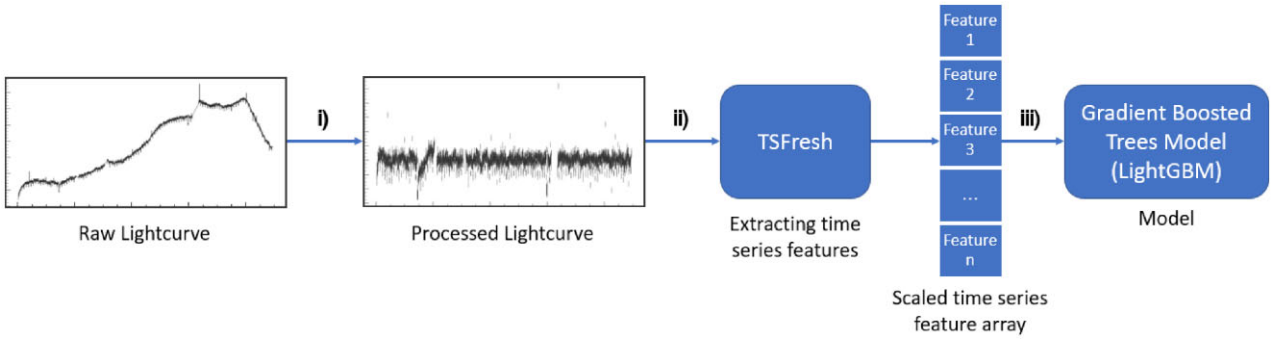

Figure 2. Workflow of our method starting from raw light curves to performing inference on the sample. In the first step (i), the raw light curves are processed to remove low-frequency variability and noise, and to sample the curve uniformly in time. In the second step (ii), features are extracted from the light curves and organized in a feature array. In the final third step (iii), the feature arrays are used as the input for a classification algorithm (gradient boosted trees) and the model is trained.

(i) processing and labelling the training data;
(ii) extracting features from each light curve using *TSFRESH*;
(iii) model training.

This workflow is shown in Fig. 2. More details on the individual steps are given in the following sections.

2.1 Preparing and labelling training data

2.1.1 Simulated data: artificially injected planets into K2 data

We obtained the K2 Campaign 7 photometry from the Mikulski Archive for Space Telescopes (MAST) and used the calibration by Vanderburg & Johnson (2014). While the processing of Vanderburg & Johnson (2014) already removed the vast majority of systematic effects, we further cleaned up the data by identifying and removing remaining cosmic ray artefacts and creating a new noise model. If a point is an outlier with respect to its neighbours, i.e. if it is more than 5σ above its previous and following point, it was assumed to be a cosmic ray artefact. Then, we removed the stellar variability that is common, depending on stellar type, with an iterative process. We smoothed the data, binned it, fit cubic splines to the binned data, clipped negative 3σ outliers and iterated this process until it converged.

Transit signals were randomly injected in half of the processed light curves in an approach similar to that introduced by Obermeier et al. (2016). We removed known planet systems from the data sample beforehand and then performed a blind search for eclipsing binary systems based on our implementation of the BLS method. As the K2 light curves sometimes had gaps, we interpolated between those gaps and sampled the time series uniformly. This was done to ensure proper feature generation using *TSFRESH*. Any light curve with a detected signal-to-noise ratio of more than 12 was removed from the sample. We then created a randomly filled space of orbital periods, stellar limb darkening and radii, planet radii and orbit inclinations. For this, we randomly simulated orbit inclinations and if at least half the simulated planet transited in front of its simulated star, we created a new injected light curve. We simulated periods between 0.23 and 30 d, which naturally lead to a bias in close-orbit planets as is the case in the real data as well. An example of the resulting simulated planet systems can be seen in Fig. 3.

Each light curve with injected planets was labelled as class '1' and remaining curves were categorized as class '0'. These labels were then used to train our classifier with the objective of identifying the injected transit signals. In total, our training set consisted of 7873 light curves out of which 3937 belong to class '1' (planet candidate). This data set was then divided into training-validation set for 10-fold cross validation (90 per cent) and test set (10 per cent).

2.1.2 Kepler data

We used the publicly available data set that was employed in the work of Shallue & Vanderburg (2018); the details about data processing can be found in section 3.2 of their paper. These light curves were produced by the Kepler pipeline (Jenkins et al. 2010), where each light curve contains around 70 000 data points equally spaced out at the interval of 29.4 min. The light curves were flattened and outliers were removed as described in the previous section. The labels for the curves were taken from the Autovetter Planet Candidate Catalog (Catanzarite 2015). The catalogue is divided into four light-curve categories: planet candidate (PC), astrophysical false positive (AFP), non-transiting phenomenon (NTP), and unknown (UNK). All UNK light curves were removed from the data set and all PC light curves were given class label 1 ('planet candidate'). Both the remaining cases were assigned to class 0 ('non-candidate'). In total, there are 3600 PC light curves and 12 137 non-PC light curves. The data are already divided into training set (80 per cent), validation set (10 per cent), and test set (10 per cent). We used the same test set for our model performance as Shallue & Vanderburg (2018) in order to compare our model performance. Apart from this, we combined their training and validation set into a training-validation set for 10-fold cross validation.

2.1.3 TESS data

For the TESS data, we again used publicly available data provided by Yu et al. (2019) for their model *AstroNet-Vetting*. Details about their light curve processing can be found in the Section 2.1, where they used the MIT Quick Look Pipeline (QLP; Huang et al. 2020) for processing the light curves. The QLP is designed to process the full-frame images (FFIs) and produces light curves using its internal calibrated images. The labelling was done by visually inspecting each light curve, categorizing them into three classes: planet candidates (PC), eclipsing binary (EB), and junk (J), where junk signals were the cases with stellar variability and instrumental noise. The data set consists of 2154 EB, 13 805 J, and 492 PC signals, where PCs were labelled as class 1 ('planet candidate') and everything else was labelled as class 0 ('non-candidate'). These data are also already divided into training set (80 per cent), validation set (10 per cent),

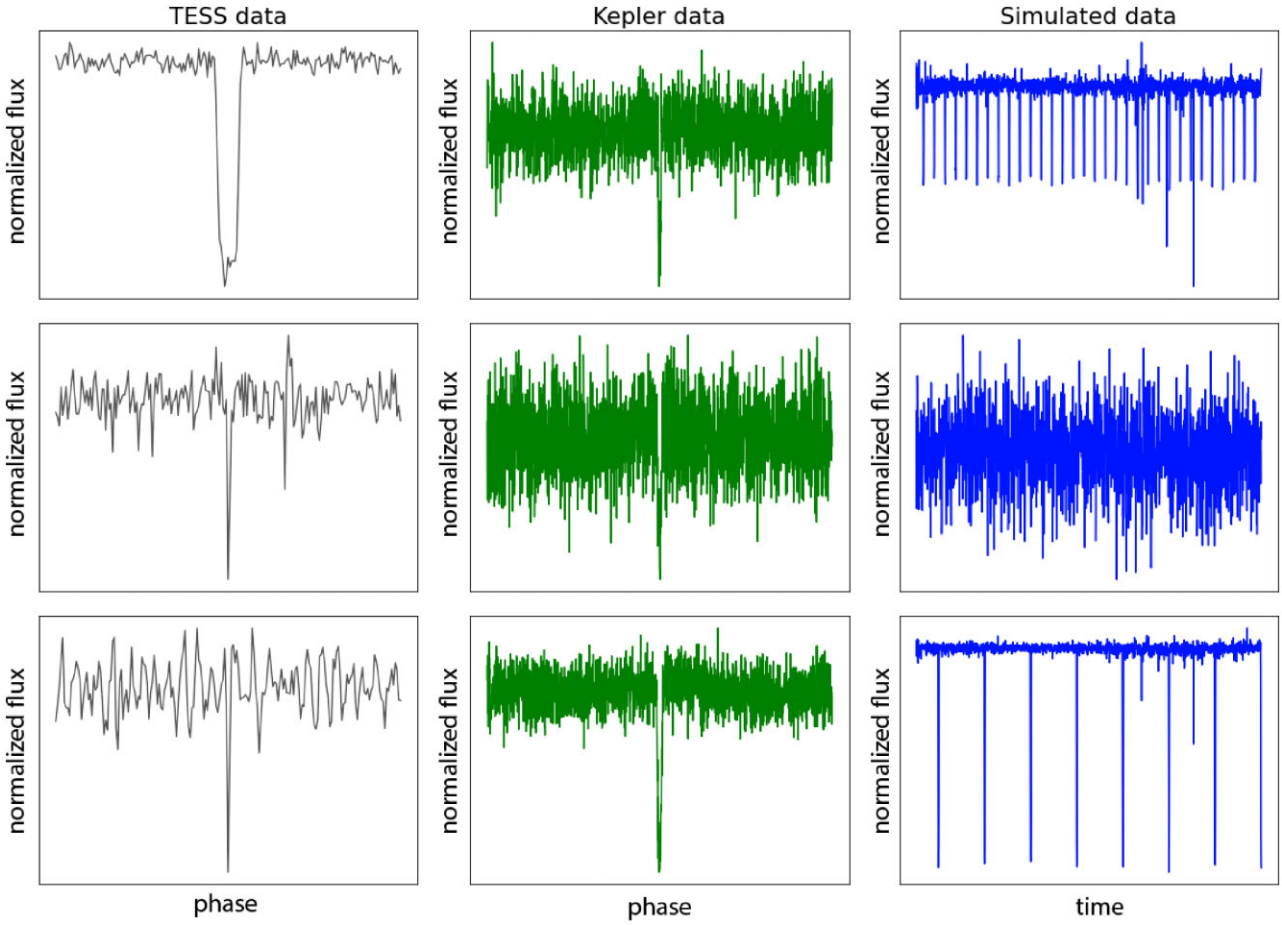

Figure 3. Arbitrary selection of normalized sample light curves containing transit signals. All of these curves are taken from our test set of the respective data set. It can be observed that the TESS [black, processing described in Yu et al. (2019)] and Kepler [green, processing described in Shallue & Vanderburg (2018)] data are a lot more noisy and likely to contain a lower number of transits. Simulated data (blue, processing described in Section 2.1.1) on the other hand is cleaner even in the unfolded state in most cases (as shown), as all the curves with S/N over 12 were removed from the data set before the planet signals were injected.
However, some of the injected transits were generated with low S/N values to imitate the realistic scenarios. The second light curve of simulated data is one such case.

and test set (10 per cent). Like the previous case, we used the same test set and combined the training and validation set into a training-validation set for 10-fold cross validation.

Some of the sample light curves containing transit signals are shown in Fig. 3. All of these curves are taken from our test set of the respective data set. It can be seen that the TESS (shown in blue) and Kepler (shown in green) data are a lot more noisy and likely to contain a lower number of transits. On the other hand, the simulated curves are cleaner as all the curves with S/N over 12 were removed from the data set before the planet signals were injected. However, some of the injected transits such as the second light curve of the simulated data were generated with low S/N values to imitate realistic scenarios.

2.2 Feature extraction

Having processed the light curves, the next step is to extract features, which capture information about the characteristics of each light curve and are used as the input for our classification model both for training and inference. We used the PYTHON framework *TSFRESH* (Christ et al. 2018) to extract features that are typical for time series such as light curves, e.g. the absolute energy of the time series (the sum over the squared values), the number of values that are higher than the mean, and the coefficients of the one-dimensional discrete Fourier transform. For all used data sets, the light curves are resampled to the frequency of 1 h, i.e. all light curves are reformatted into windows of 1 h to ensure that data points are uniformly distributed in time, a process commonly referred to as 'resampling'. Although this resampling process is not mandatory, it is a standard practice and proven to produce a better representation of the data, leading to faster convergence. In the resampling process, if multiple data points are inside one window, they are summed up and missing data points on the curves are interpolated. The frequency of 1 h is chosen to ensure minimum data loss while not increasing the data size significantly.

These resampled time series can now be directly used with multiple time series analysis tools. We used *TSFRESH*'s efficient feature extraction setting that extracted around 790 generalized time series features.[1] An exhaustive list of the primary features calculated by

[1]Details of these features can be found at: https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html

TSFRESH and their descriptions are given in Appendix A. Some of these primary features can be broken down into multiple secondary features. Moreover, these features can be calculated on different window sizes of the time series. Calculating various combinations of these primary and secondary features on different time-scales enabled us to produce a list of around 790 features. After the feature extraction process, we implemented some standard data pre-processing steps for machine learning. We removed all irrelevant features, i.e. those with values that were constant throughout the data set, and imputed missing values by interpolating. Lastly, we scaled the whole data set using a robust scaler. This data set was then used for training a tree-based classifier.

2.3 Model training

Our model is a binary classifier, i.e. it classifies every light curve into two classes, either 'planet candidate' or 'non-candidate'. We employed a gradient boosted tree (GBT) model using the popular machine learning framework *lightGBM* (Ke et al. 2017) for our classifier. A GBT model is an ensemble of decision trees which are trained in sequence. In each iteration, the GBT model is trained to reproduce the results of the decision trees by fitting the negative gradients (also known as residual errors; see Friedman 2001).

Moreover, we used a 10-fold cross validation (CV) during training. This means the training set is split into 10 smaller sets and the model is trained using nine sets at a time. The remaining last set is used to evaluate the model performance. This is done iteratively until the model has been evaluated on all 10 sets. The final performance is then the average of all the values computed in the iteration.

We use the following four metrics to evaluate our results:

(i) Accuracy: the fraction of samples correctly classified, including both 'planet candidates' and 'non-candidates'.

(ii) Precision: the fraction of predicted 'planet candidates' that are true planets, i.e. the accuracy of the 'planet candidate' predictions.

(iii) Recall: ratio of the true planets correctly detected by the classifier as 'planet candidate', i.e. what fraction of true planets is recovered (also called 'true positive rate').

(iv) AUC: the area under the receiver operating characteristic (ROC) curve, called the AUC, gives the probability that a randomly chosen light curve with a true planet is ranked higher than a randomly chosen light curve without a planet.

The accuracy is generally not a proper metric to judge the performance of the planet detection algorithm, since most of the exoplanet detection data sets are unbalanced, i.e. usually there are significantly more light curves without any planet signal than cases with a planet. For instance, in the TESS data set only 3 per cent of the light curves are true planet candidates. If a classifier simply predicts 'non-candidate' for all light curves it will still lead to a 'high' accuracy of 97 per cent, even though it has not learned from the data and cannot identify any planet candidates.

While a high precision ensures that most of the predicted 'planet candidates' are true planet candidates, it is also not a very useful metric, as a trivial way to achieve a high precision is to make only few 'planet candidate' predictions and ensure they are correct. Consequently, many possible planet candidates would be missed. As we would rather allow for a higher number of false positives than losing possible planet signals, the recall is a much more important metric to assess the performance of our algorithm.

A model with a high recall may lead to a lower precision and vice versa, which is commonly known as the precision-recall trade-off. Classification decisions are based on a decision threshold, which is another hyperparameter of the model. If the predicted model probability surpasses this threshold, the light curve is classified as 'planet-candidate'. For a typical classification problem, a threshold of $p_{\rm threshold} = 0.5$ is chosen, so that light curves with a predicted probability of more than 0.5 are considered 'planet candidates'. If a higher decision threshold is chosen, the model will have a higher precision but a lower recall, and vice versa, so it can be optimized to yield a high recall.

Finally, the AUC is independent of the decision threshold, as it gives the integral over all possible decision thresholds. Therefore, it can be used to optimize the remaining hyperparameters first to get a model that performs well for any threshold. In the second step, the threshold can then be chosen to achieve a high recall. We argue that out of the above four metrics AUC and recall are the most important as the main imperative is to design a method that is able to recover a high percentage of light curves containing planet signals. In other words, we aim to optimize our model for not missing light curves that contain a true planet signal.

Due to these reasons, we first optimized all hyperparameters of our model to maximize the AUC. This optimization was performed only on the cross-validation set. Then we optimized the decision threshold to obtain a high recall while maintaining precision as high as possible. In practice, once the all hyperparameters except for the threshold are fixed, we check whether it is possible to get both precision and recall in the upper percentile (>90 per cent), and if so select the highest possible recall while maintaining a precision in that percentile. If this is not possible, we allow the precision to drop to the next percentile and select the highest possible recall. Once the parameters are optimized, results are evaluated on the test set.

3 RESULTS

3.1 Simulated data

In order to attain a proof of concept, the method was first applied to our simulated data that was obtained as described in Section 2.1.1 and consists of 7873 light curves. Extracting features for each of the light curves and using them as inputs for the GBTs, the model outputs the probability for a light curve to contain a transit. Given a decision threshold, the light curve is then either classified as 'planet-candidate' or 'non-candidate'. Fig. 4 shows the precision, recall, and the F1 score (the harmonic mean between precision and recall), as a function of the decision threshold. All curves have been computed with the model that was optimized for a high AUC as described above. Choosing a decision threshold of $p_{\rm threshold} = 0.13$, we get a recall of 0.94 and precision of 0.92. This result is preferred over the one with the standard threshold of 0.5 as for planet detection, it is preferred to extract the maximum possible light curves at the cost of a minor increase in false positives. The results on our validation set are shown in Table 1.

We then used the BLS algorithm on our simulated data set. For this, we searched for periods between 0.23 and 30 d, identical to the simulations. The test periods of BLS were evenly sampled in the 1/p space and were chosen such that the periods were distributed more densely than the typical exposure times even at the highest period. In order to speed up the signal search, we binned the folded light curves into 500 bins per test period, as in Obermeier et al. (2016). We were able to detect more than 96 per cent of the true planets as opposed to 94 per cent with our machine learning based method. While it is not surprising for BLS to reach a higher success rate, about 12 per cent of the sample had an S/N between 4.5 and 12, making them pretty marginal detections that would likely be disregarded in the course

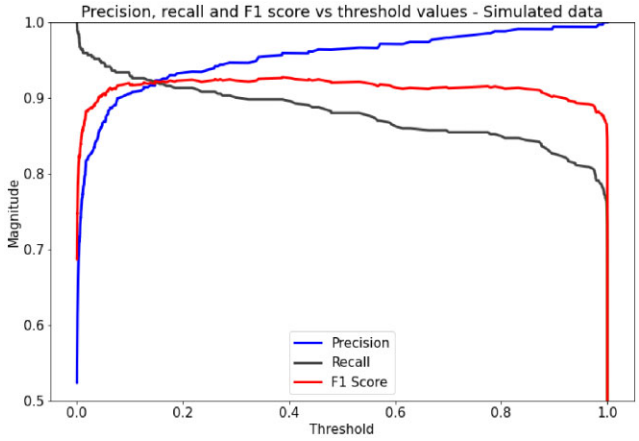

Figure 4. Precision, recall, and F1 score (harmonic mean of precision and recall) as function of the decision threshold for the test set of our simulated data. The typical threshold of 0.5 for classification problems can be adapted to increase or decrease the model sensitivity. For the simulated planet data set, choosing a threshold of 0.13 results in a recall of 0.94 and a precision of 0.92. As the objective is to retrieve the maximum number of possible light curves containing transit signals, this result is preferred over the ones produced by a default threshold of 0.5. The AUC of the model is 0.92.

Table 1. Results on simulated data.

AUC	Recall	Precision	Accuracy
0.92	0.94	0.92	0.91

of a transit survey, as an S/N of 12 was our previous cut-off for the generation of the transit-free data set. Overall, about 73 per cent of the BLS results had a robust S/N of above 30 and 53 per cent even had an S/N of above 100.

When analysing the low S/N or wrong-period detections of BLS, we found a curious grouping in low S/N detections around a simulated period of 0.24–0.25 d that may correspond to thruster firings of K2. We also discovered an overlap in the cases that were not detected by our method and BLS. We investigated these cases manually and found that in this sample the transits were injected randomly with random parameters which resulted in many non-detectable transits such as:

(i) Cases with a relatively low inclination angle for the given star, which resulted in very small planet signals.
(ii) Cases with low S/N ratio where the injected transit signal was weaker than the noise, and hence it was not detected either by our method or by BLS.

The majority of these cases had a signal-to-noise ratio of S/N < 12, so that the signal was simply too weak. These results demonstrate that our machine learning based method can detect planets more accurately and efficiently than BLS, especially in cases with low signal-to-noise ratio.

Similar results were also shown by Pearson, Palafox & Griffith (2018), who compared various machine learning methods with BLS. Their machine learning models were able to detect planet signals with higher signals and much lower false positive rate compared to BLS. Therefore, our next step was to apply our methodology on realistic data sets, i.e. the Kepler and TESS light curves.

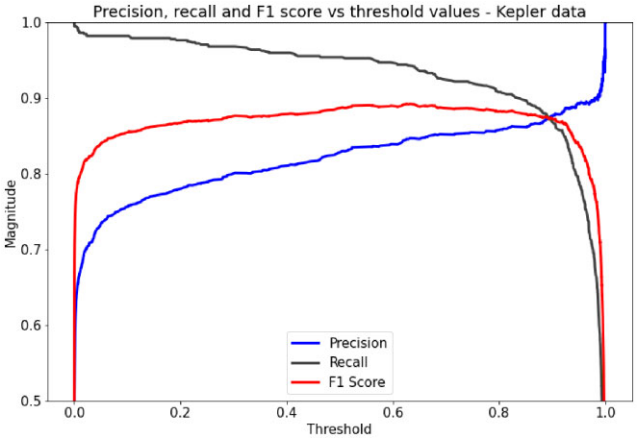

Figure 5. As Fig. 4, but for the Kepler data set. Choosing a threshold of 0.46 results in a recall of 0.96 and a precision of 0.82. The AUC of the model is 0.948.

Table 2. Results on Kepler data.

Type	AUC	Recall	Precision
Shallue & Vanderburg (2018)	0.988	0.95	0.93
Our method	0.948	0.96	0.82

3.2 Kepler data

Shallue & Vanderburg (2018) introduced *Astronet*, a deep learning model that produced very promising results on the Kepler data. Shortly after, Ansdell et al. (2018) introduced an improved version of *Astronet* that produced a higher average precision. To date, these are the best results on this data set. However, Ansdell et al. (2018) did not provide AUC in their work and since we used AUC to optimized and evaluate our model, we cannot directly compare their results in this paper. Hence, we used *Astronet* as a benchmark to compare our results.

They shared their training data set publicly, which we used to evaluate our method, and we used their results as a benchmark for our results. The data set consists of 15 737 light curves that were labelled into three classes by the Autovetter and combined into a 'planet candidate' class and a 'non-candidate' class, as described in Section 2.1.2. We again extracted features from the light curves with *TSFRESH* and used them along with the class labels to train a GBT model. Optimizing the hyperparameters, we found the best model to yield an AUC of 0.948. The resulting precision, recall, and F1 score as a function of the decision threshold are shown in Fig. 5.

As discussed in Section 2.3, we decided to optimize the threshold to obtain the highest possible recall, while maintaining a good precision. Choosing a threshold of $p_{\text{threshold}} = 0.46$ results in a recall of 0.96 and a precision of 0.82. A comparison of the results of our method with the results by Shallue & Vanderburg (2018) is summarized in Table 2. It can be seen that the results of both the methods are comparable. While our model has a somewhat lower AUC, it has a slightly higher recall, albeit at a considerably lower precision. However, as discussed before, we specifically aimed at getting a high recall instead of precision, as on major criterion is to identify as many planet candidates as possible.

To further analyse the Kepler data set and to visualize how different light curves cluster, we embedded the data in a low-dimensional space of two dimensions with the t-distributed stochastic neighbour embedding (t-SNE; Maaten & Hinton 2008) algorithm. To this end,

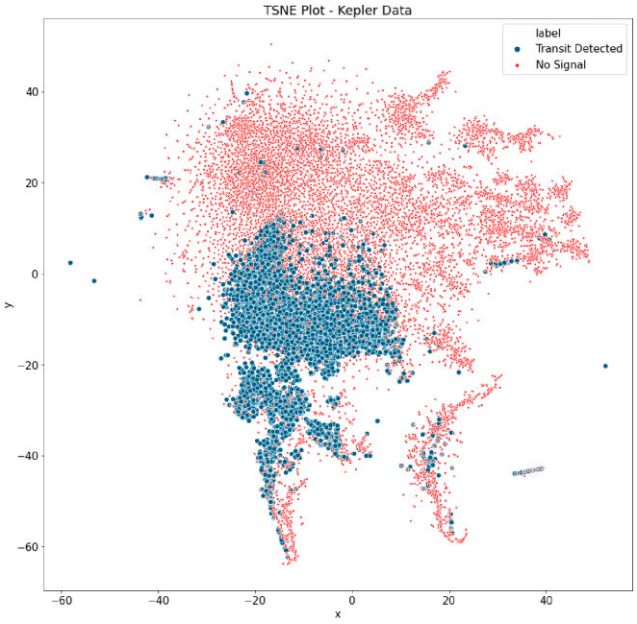

Figure 6. t-SNE visualization of the Kepler data. Light curves with planet transit signal(s) are shown in blue. It can be observed that these curves are clustered in one region of the space.

we extracted time series features from each light curves as described in Section 2.2. These features were then scaled using a robust scaler, and any missing values were filled using the mean of that column, which resulted in around 700 processed features. The top 70 per cent of those features were selected, based on the feature importance found by the GBT model (as using only the top 70 per cent of the features resulted in an optimized AUC), and used as the input for the t-SNE algorithm for dimensionality reduction. The results are presented in Fig. 6.

As the t-SNE algorithm retains the local structure of the data, similar light curves are modelled by nearby points with high probability, i.e. the points closer in the 2D representation are also closer in the high-dimensional space. However, a point far away in the 2D space might not be far away in high-dimensional space. As Fig. 6 shows, most of the light curves containing a transit are clustered in one region of the plot, which further validates our pre-processing and feature extraction process. Broadly, there are three main clusters in the plot, and samples from these clusters can be analysed to understand how these clusters vary from each other. However, we left this analysis for future investigations.

3.3 TESS data

TESS light curves are in many way different from the Kepler data set. Kepler observed a fixed field of view in its 4 yr timeline and K2 observed each of its 19 campaigns for 80 d. On the other hand, TESS observed each sector for 27 d. Longer baselines contain more details and lead to a much higher signal to noise in terms of detections, hence the extracted features are also more rich in information. The short time span of TESS implies that each light curve (and features extracted from it) tends to contain less data points with transits. Consequently, it becomes more difficult for a classifier to differentiate a case with a planet signal from a case without it. Moreover, the short length of the light curves makes the presence of multiple-periodic transit signals less likely to be observed. For longer exoplanet transits, this problem is further compounded if only a single transit is recorded. In contrast, multiple transiting planets in the Kepler data may lead to an automatic confirmation due to the low probability of any fitting false-positive scenario.

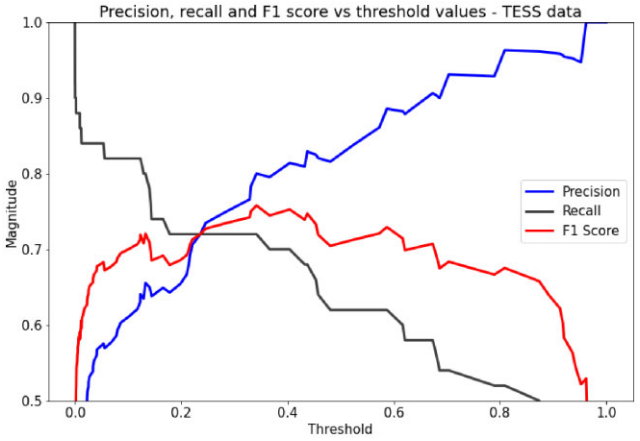

Figure 7. As Fig. 4, but for the TESS data set. With a threshold of 0.12 the recall is 0.82 and the precision is 0.63. The AUC of the model is 0.80.

Table 3. Results on TESS data with default threshold.

Type	AUC	Recall	Precision
Yu et al. (2019)	0.98	0.57	0.65
Our method	0.81	0.62	0.84

Table 4. Results on TESS data with optimized threshold.

Type	AUC	Recall	Precision
Yu et al. (2019)	0.98	0.89	0.45
Our method	0.81	0.82	0.63

The first machine leaning classifier to be trained and tested on real TESS data was introduced by Yu et al. (2019). Their model is an adapted version of Shallue & Vanderburg (2018) as shown in fig. 2 of their paper. We again trained our model on the data set publicly shared by Yu et al. (2019) for their AstroNet-Vetting network and used their vetting results as benchmark. The data consists of 16 500 light curves, with only 490 planet candidates. Applying *TSFRESH* and a GBT model, we optimized the hyperparameters to get an AUC of 0.81. The resulting precision, recall, and F1 score as a function of the decision threshold are shown in Fig. 7. We again optimized the decision threshold to get a high recall, which resulted in a threshold of $p_{\text{threshold}} = 0.12$, a recall of 0.82, and a precision of 0.63 for the 'planet candidate' class. A comparison of the results of both the methods with default and optimal thresholds is shown in Tables 3 and 4, respectively.

With the optimized threshold, our model was able to find 40 out of 49 curves with planet transit in our test set as opposed to 44 samples identified by Yu et al. (2019) on this test set. On the other hand, the precision of our model is significantly higher, i.e. it will result in almost half of the false positives as encountered by Yu et al. (2019). With the default threshold, the deep learning method by Yu et al. (2019) resulted in a much higher AUC than our model, but only had a recall of 0.57, so only about 57 per cent of all planets were detected by the model. On the other hand, our machine learning method is able to obtain a recall of 0.62.

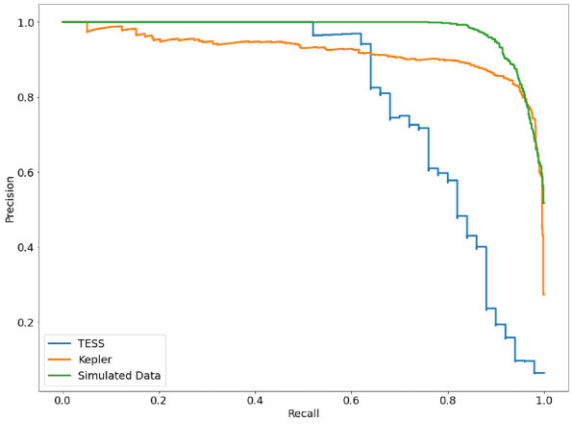

Figure 8. Precision as a function of recall for the test sets of TESS, Kepler, and the simulated data, demonstrating the trade-off between precision and recall for different decision thresholds. The performance for the simulated and the Kepler data is similar, with a slight increase in the number of false positives for the Kepler data set, as it is more noisy. The performance for the TESS data set is worse as a result of a high class imbalance.

Of course, a recall of 0.82 is still not ideal and there is a lot more work to be done before such system can be used in production, both in preparing the data set and tuning the hyperparameters. For planet detection with both the Kepler and the TESS data sets, class imbalance is one of the biggest problems the model is facing. Now that more and more planets are identified in the TESS data, more light curves with planets can be gathered and used to train machine learning models, which is expected to improve the results (see the discussion in Section 4). Even though our machine learning algorithm is not yet fit for production with the TESS data, it is certainly possible to use the model for reliably eliminating false positives, so that the amount of human vetting can be greatly reduced on cases that are a clear false positive.

The performance for all three data sets is compared in Fig. 8, where we plot the precision as a function of recall for each data set. We can see that the performance for the simulated and the Kepler data is similar. There is a slight increase in the number of false positives in the Kepler data set, as it is more noisy than the simulated one. On the other hand, the performance for the TESS data set is significantly worse, with the high class imbalance being the primary cause of this. With a more balanced data set, the performance for the TESS data is expected to approach the other two cases.

4 DISCUSSION

4.1 Classical machine learning versus deep learning

One of the primary differences between classical machine learning and deep learning methods is the fact that deep learning models are able to automatically extract useful features from raw data, while classical machine learning methods usually cannot deal with raw data and need extracted features. Consequently, the classical machine learning approach does not work well when problems become very complex, such as language translation or image classification, where instead deep learning is pushing new frontiers. However, classical machine learning methods should not be disregarded, as they are much more efficient and can still produce valuable results for several problems, such as classifying quasars (Carrasco et al. 2015), or identifying asteroids (Smirnov & Markov 2017).

Our classical machine learning approach has the following advantages over state-of-the-art deep learning methods:

(i) It can work with only a global view of the light curve, whereas deep learning additionally requires folded or secondary views.
(ii) Training is less time consuming and takes less than 5 min to train on a 2 CPU system. This also indicates that it can be quickly adapted to a new data source, while deep learning can take more than 5 h for training and much longer for tuning hyperparameters.
(iii) The exact same model set-up and code can be used for data from different sources such as Kepler, K2, and TESS. Hyperparameters need to be optimized only once, whereas deep learning almost always needs re-tuning when the data set changes.
(iv) The most important features can be automatically identified, which allows for a better understanding of the data and the underlying physical processes. By contrast, deep learning models are usually black boxes, so that it is hard to interpret the results.
(v) No special hardware, such as a GPU, is required for training.

However, our approach has the following disadvantages compared to state-of-the-art deep learning methods:

(i) Generally, the performance is lower compared to deep learning, which when properly trained can often achieve better results.
(ii) Time series data such as the global view, and folded view can be directly used as an input for deep learning, while a classical machine learning model requires extracted features from the time series data.

4.2 The vetting tool

Machine learning methods are still relatively new in astrophysics and often met with scepticism. While there is a new paradigm of building classification techniques with machine learning methods, it still has a long way to go before being used in production pipelines. One major factor is the non-deterministic or black-box nature of such methods. Unlike conventional algorithms, the outcome of such methods cannot always be understood due to the non-linear and stochastic nature of such methods, so that it is harder for the user to explain the reason behind predictions made by the model. Even though these models can outperform conventional methods like BLS, they do not always make correct predictions that is another contributing factor for the scepticism. Hence, it is important to note that they cannot completely replace human vetting experts. However, if they are supported to be used alongside the domain knowledge, these techniques can help to automate processes like planet detection and enable us to deal with the growing data size in astronomy.

These tools are not widely used yet but for these systems to progress further, it is important to take feedback and expertise from the community on the methods. For this reason, we have developed a vetting tool where a few results from our model can be explored in an interactive way without any knowledge of the underlying machine learning method or the feature calculation technique. The tool is hosted on GitHub.[2]

The vetting tool allows the user to explore cases where the model assigns a high probability of planet candidacy. It allows the user to view the light curve in the global view and the folded view (also called local view). It further gives a list of features that were important in the model prediction for that case. It can be easily adapted to make model inference on any new light curve.

[2]https://github.com/abhmalik/Exoplanet-Vetting-Tool

4.3 Future steps

With this paper, we tried to provide a new direction for a light and efficient automatic vetting system. However, there is a lot more that can be done in this direction.

Since our method is able to work with data from different sources such as Kepler, K2, and TESS, it is possible to construct a global classifier that can process light curves of any length from any data source directly. This would enable us to continue using the same model in case the data format changes, as happened for K2 when light curves had a time gap.

As we saw, the performance of our model was worse for the TESS data set compared to the other two data sets. The primary reason for this was the high class imbalance in TESS data, i.e. less than 3 per cent of the light curves contain transit signals, and most of them are just single transits. Now that more and more planets are being confirmed from TESS data, it will be possible to construct a data set with better class balance, which might help us to reach the performance levels we are reaching with the Kepler and the simulated data.

As shown in the t-SNE plot in Fig. 6, light curves are clustered in different regions. We can extract samples from these groups and analyse how the light curves in different group differ. This information can further be used to optimize the model performance for that particular data set.

Another possibility would be to forego the supervised learning approach and turn towards a semisupervised approach. To this end the data would first have to be embedded in a lower dimensional space (e.g. using an autoencoder or principal component analysis). An unsupervised clustering algorithm could then find clusters of data in the second step. Finally, a limited amount of well-understood and labelled light curves could be used to label the individual clusters. If a new light curve would then be determined to be a member of a specific cluster, the cluster label can thus be used for the light curve. We leave such an approach for future work.

5 CONCLUSIONS

Machine learning methods have seen very active development in the last decade and now they are an essential part of our way of working. In fact, for everyone who interacts with a computer or smartphone, it is highly likely to interact with some sort of machine learning programme. They are also widely used in sciences for several use cases such as detecting diseases (Sajda 2006) and finding string models in a string theory landscape (Deen et al. 2020). Within astronomy, there have been many applications such as classifying galaxy morphology (Wu et al. 2019), creating new large-scale structure samples without long simulations (Mustafa et al. 2019), and identifying gravitational waves (George & Huerta 2018) and gravitational lenses (Lanusse et al. 2018).

With new and advanced telescopes, data in astronomy are growing at a fast pace. Conventional methods that involve human judgements are not efficient and prone to variability depending on the investigating expert. For example, the commonly used method for exoplanet detection, BLS, produces large number of false positives in case of noisy data, which have to be reviewed manually.

In this paper, we proposed a novel planet detection method based on classical machine learning. Our method consists of automatically extracting time series features from light curves which are then used as input to a gradient boosted trees model. We were able to demonstrate that with our machine learning method, we could identify light curves with planet signals more accurately as compared to BLS while significantly reducing the number of false positives.

Pioneering work in the area has been done by Shallue & Vanderburg (2018) using deep learning. Although deep learning methods often outperform classical machine learning approaches, specifically in complex problems such as machine translation and object detection, they tend to be harder to optimize and usually require a long training time on special hardware (GPUs). Moreover, they typically rely on large sets of training data and are more difficult to hypertune as they have to navigate a much bigger parameter space as compared to classical machine learning models. Hence, in this work, we attempted to move away from the standard deep learning approach and introduced a new direction that provides a light-weight model.

Our approach consists of three main steps. First, the raw light curves were processed, so that systematic effects, cosmic rays, and stellar variability were removed to get flat curves with a uniform time sampling. In the second step, the processed light curves were then analysed by the *TSFRESH* library, which extracted features that are typical for time series (e.g. absolute energy and number of values above mean). Finally, the third step was to use these features to train a gradient boosted tree model to predict the class ('planet candidate' or 'non-candidate') for each light curve. In this way, our model is able to distinguish important features from unimportant ones, and there is no need for manually selecting features. Moreover, our approach does not require special hardware like GPUs and it takes about 2 min to train our classical machine learning model on a dual core CPU system. This means it can be easily trained even on low end computers and can be revised quickly in case it is required (e.g. when new data are available).

We found that our machine learning method is able to compete with the classical method to identify planet signals, BLS. For our simulated data set, BLS was able to identify 96 per cent of true planets, while the machine learning model could identify 94 per cent. Moreover, there is a significant advantage in CPU time compared to classical detection methods, as our machine learning method can be fully executed in a matter of minutes, using only two CPU cores, which would have taken days for a comparable BLS run on such a system. We also compared the performance of our results with other state-of-the-art models based on deep learning, and found that, for the Kepler data, our model is able to achieve comparable results (a recall of 0.96 at a precision of 0.82). For the TESS data our model is able to achieve a recall comparable to the deep learning models (82 versus 89 per cent), but at a precision that is significantly higher (81 per cent versus 44 per cent), so that fewer false positives have to be removed manually.

While these results are very encouraging, such systems are not yet robust enough to be broadly applicable. Even if a machine learning model works well in the development environment, it is prone to make mistakes on unseen data. Hence, such methods should be used alongside with human supervision. None the less, at the current stage, these models can provide a very reliable system to rule out large number of false positives and can drastically reduce the number of cases requiring manual reviews.

ACKNOWLEDGEMENTS

We are grateful to Arno Riffeser and Jana Steuer for useful discussions and comments. We would also like to thank Christopher Shallue and Liang Yu for publicly sharing their code and training data. This enabled us to quickly test our model on different data sets and provided a benchmark to compare our results. We thank the

developers of LIGHTKURVE (Lightkurve Collaboration 2018), NUMPY (Walt, Colbert & Varoquaux 2011), MATPLOTLIB (Hunter 2007), SCIKIT-LEARN (Pedregosa et al. 2011), and TENSORFLOW (Abadi et al. 2015) for their very useful free software. The Astrophysics Data Service (ADS) and arXiv preprint repository were used intensively in this work. BPM acknowledges an Emmy Noether grant funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – MO 2979/1-1.

DATA AVAILABILITY

The derived data in this article will be shared on reasonable request to the corresponding author.

REFERENCES

Abadi M. et al., 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, Available at: https://www.tensorflow.org/
Abrishami S., 2019, PhD thesis, Florida State University
Ansdell M. et al., 2018, ApJ, 869, L7
Batalha N. M. et al., 2011, ApJ, 729, 27
Beaulieu J. P. et al., 2006, Nature, 439, 437
Benedict G. F. et al., 2002, ApJ, 581, L115
Borucki W. J. et al., 2010, Science, 327, 977
Burke C. J. et al., 2015, ApJ, 809, 8
Campbell B., Walker G. A. H., Yang S., 1988, ApJ, 331, 902
Carrasco D. et al., 2015, A&A, 584, A44
Catanzarite J. H., 2015, Technical report, Autovetter Planet Candidate Catalog for Q1-Q17 Data Release 24
Charbonneau D., Brown T. M., Latham D. W., Mayor M., 2000, ApJ, 529, L45
Chaushev A. et al., 2019, MNRAS, 488, 5232
Chauvin G., Lagrange A. M., Dumas C., Zuckerman B., Mouillet D., Song I., Beuzit J. L., Lowrance P., 2004, A&A, 425, L29
Christ M., Braun N., Neuffer J., Kempa-Liehr A. W., 2018, Neurocomputing, 307, 72
Coughlin J. L. et al., 2016a, ApJS, 224, 12
Coughlin J. L. et al., 2016b, ApJS, 224, 12
Dattilo A. et al., 2019, AJ, 157, 169
Deen R., He Y.-H., Lee S.-J., Lukas A., 2022, Phys. Rev. D, 105, 046001
Fressin F. et al., 2013, ApJ, 766, 81
Friedman J. H., 2001, Ann. Stat., 29, 1189
George D., Huerta E. A., 2018, Phys. Lett. B, 778, 64
Gilbert E. A. et al., 2020, AJ, 160, 116
Howell S. B. et al., 2014, PASP, 126, 398
Huang C. X. et al., 2018, ApJ, 868, L39
Huang C. X. et al., 2020, RNAAS, 4, 204
Hunter J. D., 2007, Comput. Sci. Eng., 9, 90
Jenkins J. M. et al., 2010, ApJ, 713, L87
Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T.-Y., 2017, in Guyon I., Luxburg U. V., Bengio S., Wallach H., Fergus R., Vishwanathan S., Garnett R., eds, Advances in Neural Information Processing Systems 30. Curran Associates, Inc., Long Beach CA, US, p. 3146
Khan I., Choi S., Kwon Y.-W., 2020, Sensors, 20, 800
Kovács G., Zucker S., Mazeh T., 2002, A&A, 391, 369
LaCasse P. M., Otieno W., Maturana F. P., 2019, Appl. Sci., 9, 853
Lanusse F., Ma Q., Li N., Collett T. E., Li C.-L., Ravanbakhsh S., Mandelbaum R., Póczos B., 2018, MNRAS, 473, 3895
Latham D. W., Mazeh T., Stefanik R. P., Mayor M., Burki G., 1989, Nature, 339, 38
Lightkurve Collaboration, 2018, Astrophysics Source Code Library, record ascl:1812.013
Maaten L. v. d., Hinton G., 2008, J. Mach. Learn. Res., 9, 2579
Mao S., Paczynski B., 1991, ApJ, 374, L37
Mayor M., Queloz D., 1995, Nature, 378, 355
McCauliff S. D. et al., 2015, AJ, 806, 6
Mislis D., Bachelet E., Alsubai K., Bramich D., Parley N., 2016, MNRAS, 455, 626
Moore P. J., Lyons T. J., Gallacher J., 2019, PLOS One, 14, 1
Muirhead P. S. et al., 2012, ApJ, 747, 144
Mustafa M., Bard D., Bhimji W., Lukić Z., Al-Rfou R., Kratochvil J. M., 2019, Comput. Astrophys. Cosmol, 6, 1
Muterspaugh M. W. et al., 2010, AJ, 140, 1657
Naef D. et al., 2001, A&A, 375, L27
Obermeier C., 2016, PhD thesis, Ludwig-Maximilians-Universität München
Obermeier C. et al., 2016, A&A, 587, A49
Pearson K. A., Palafox L., Griffith C. A., 2018, MNRAS, 474, 478
Pedregosa F. et al., 2011, J. Mach. Learn. Res., 12, 2825
Ricker G. R. et al., 2015, JATIS, 1, 014003
Sajda P., 2006, Annu. Rev. Biomed. Eng., 8, 537
Shallue C. J., Vanderburg A., 2018, AJ, 155, 94
Smirnov E. A., Markov A. B., 2017, MNRAS, 469, 2024
Thalmann C. et al., 2009, ApJ, 707, L123
Vanderburg A., Johnson J. A., 2014, PASP, 126, 948
Vanderspek R. et al., 2019, ApJ, 871, L24
Walt S. v. d., Colbert S. C., Varoquaux G., 2011, Comput. Sci. Eng., 13, 22
Wolszczan A., Frail D. A., 1992, Nature, 355, 145
Wu C. et al., 2019, MNRAS, 482, 1211
Wu J., Zhou T., Li T., 2020, Entropy, 22, 140
Yu L. et al., 2019, AJ, 158, 25

APPENDIX A: *TSFresh* PRIMARY FEATURE LIST

An exhaustive list of the primary features calculated by *TSFresh* and their descriptions are given in Table A1.

Table A1. List of all the primary features and their description calculated by *TSFresh*.

Feature type	Description
abs_energy (x)	Returns the absolute energy of the time series which is the sum over the squared values
absolute_maximum (x)	Calculates the highest absolute value of the time series x
absolute_sum_of_changes (x)	Returns the sum over the absolute value of consecutive changes in the series x
agg_autocorrelation (x, *param*)	Descriptive statistics on the autocorrelation of the time series.
agg_linear_trend (x, *param*)	Calculates a linear least-squares regression for values of the time series that were aggregated over chunks versus the sequence from 0 up to the number of chunks minus one.
approximate_entropy (x, m, r)	Implements a vectorized Approximate entropy algorithm.
ar_coefficient (x, *param*)	This feature calculator fits the unconditional maximum likelihood of an autoregressive AR (k) process.
augmented_dickey_fuller (x, *param*)	Does the time series have a unit root?
autocorrelation (x, *lag*)	Calculates the autocorrelation of the specified lag
benford_correlation (x)	Useful for anomaly detection applications
binned_entropy (x, max_bins)	First bins the values of x into max_bins equidistant bins
c3 (x, *lag*)	Uses c3 statistics to measure non linearity in the time series
change_quantiles (x, ql, qh, $isabs$, f_agg)	First fixes a corridor given by the quantiles ql and qh of the distribution of x
cid_ce (x, *normalize*)	This function calculator is an estimate for a time series complexity. A more complex time series has more peaks, valleys etc.
count_above (x, t)	Returns the percentage of values in x that are higher than t
count_above_mean (x)	Returns the number of values in x that are higher than the mean of x
count_below (x, t)	Returns the percentage of values in x that are lower than t
count_below_mean (x)	Returns the number of values in x that are lower than the mean of x
cwt_coefficients (x, *param*)	Calculates a Continuous wavelet transform for the Ricker wavelet, which is also known as the 'Mexican hat wavelet'
energy_ratio_by_chunks (x, *param*)	Calculates the sum of squares of chunk i out of N chunks expressed as a ratio with the sum of squares over the whole series
fft_aggregated (x, *param*)	Returns the spectral centroid (*mean*), variance, skew, and kurtosis of the absolute Fourier transform spectrum
fft_coefficient (x, *param*)	Calculates the fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast
first_location_of_maximum (x)	Returns the first location of the maximum value of x
first_location_of_minimum (x)	Returns the first location of the minimal value of x
fourier_entropy (x, *bins*)	Calculate the binned entropy of the power spectral density of the time series (using the Welch method)
friedrich_coefficients (x, *param*)	Coefficients of polynomial h (x), which has been fitted to
has_duplicate (x)	Checks if any value in x occurs more than once
has_duplicate_max (x)	Checks if the maximum value of x is observed more than once
has_duplicate_min (x)	Checks if the minimal value of x is observed more than once
index_mass_quantile (x, *param*)	Calculates the relative index i of time series x where q% of the mass of x lies left of i
kurtosis (x)	Returns the kurtosis of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2)
large_standard_deviation (x, r)	Does time series have large standard deviation?
last_location_of_maximum (x)	Returns the relative last location of the maximum value of x
last_location_of_minimum (x)	Returns the last location of the minimal value of x
lempel_ziv_complexity (x, *bins*)	Calculate a complexity estimate based on the Lempel-Ziv compression algorithm
length (x)	Returns the length of x
linear_trend (x, *param*)	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one
linear_trend_timewise (x, *param*)	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one
longest_strike_above_mean (x)	Returns the length of the longest consecutive subsequence in x that is bigger than the mean of x
longest_strike_below_mean (x)	Returns the length of the longest consecutive subsequence in x that is smaller than the mean of x
matrix_profile (x, *param*)	Calculates the 1-D Matrix Profile and returns Tukey's Five Number Set plus the mean of that Matrix Profile
max_langevin_fixed_point (x, r, m)	Largest fixed point of dynamics :math:argmax_x h (x)=0' estimated from polynomial h (x)
maximum (x)	Calculates the highest value of the time series x
mean (x)	Returns the mean of x
mean_abs_change (x)	Average over first differences
mean_change (x)	Average over time series differences
mean_n_absolute_max (x, number_of_maxima)	Calculates the arithmetic mean of the n absolute maximum values of the time series
mean_second_derivative_central (x)	Returns the mean value of a central approximation of the second derivative
median (x)	Returns the median of x
minimum (x)	Calculates the lowest value of the time series x
number_crossing_m (x, m)	Calculates the number of crossings of x on m

Table A1 – *continued*

Feature type	Description
number_cwt_peaks (*x*, *n*)	Number of different peaks in x
number_peaks (*x*, *n*)	Calculates the number of peaks of at least support n in the time series x
partial_autocorrelation (*x*, *param*)	Calculates the value of the partial autocorrelation function at the given lag
percentage_of_reoccurring_datapoints_to_all_datapoints (*x*)	Returns the percentage of non-unique data points
percentage_of_reoccurring_values_to_all_values (*x*)	Returns the percentage of values that are present in the time series more than once
permutation_entropy (*x*, *tau*, *dimension*)	Calculate the permutation entropy
quantile (*x*, *q*)	Calculates the q quantile of x
query_similarity_count (*x*, *param*)	This feature calculator accepts an input query subsequence parameter, compares the query (under z-normalized Euclidean distance) to all subsequences within the time series, and returns a count of the number of times the query was found in the time series (within some predefined maximum distance threshold)
range_count (*x*, *min*, *max*)	Count observed values within the interval [min, max)
ratio_beyond_r_sigma (*x*, *r*)	Ratio of values that are more than r $*$ std (x) away from the mean of x
ratio_value_number_to_time_series_length (*x*)	Returns a factor which is 1 if all values in the time series occur only once, and below one if this is not the case
root_mean_square (*x*)	Returns the root mean square (*rms*) of the time series
sample_entropy (*x*)	Calculate and return sample entropy of x
set_property (*key*, *value*)	This method returns a decorator that sets the property key of the function to value
skewness (*x*)	Returns the sample skewness of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1)
spkt_welch_density (*x*, *param*)	This feature calculator estimates the cross power spectral density of the time series x at different frequencies
standard_deviation (*x*)	Returns the standard deviation of x
sum_of_reoccurring_data_points (*x*)	Returns the sum of all data points, that are present in the time series more than once
sum_of_reoccurring_values (*x*)	Returns the sum of all values, that are present in the time series more than once
sum_values (*x*)	Calculates the sum over the time series values
symmetry_looking (*x*, *param*)	Boolean variable denoting if the distribution of x looks symmetric
time_reversal_asymmetry_statistic (*x*, *lag*)	Returns the time reversal asymmetry statistic
value_count (*x*, *value*)	Count occurrences of value in time series x
variance (*x*)	Returns the variance of x
variance_larger_than_standard_deviation (*x*)	Is variance higher than the standard deviation?
variation_coefficient (*x*)	Returns the variation coefficient (standard error/mean, give relative value of variation around mean) of x

This paper has been typeset from a TEX/LATEX file prepared by the author.