Hunting for Exoplanets using Machine Learning

Federico Huxhagen

January 2021

Overview

The search for planetary systems across the universe has been one of the most active and prolific areas of scientific research in the last few decades. Since the discovery of the first exoplanet in 1995, a substantial amount of progress has been made in this field, with over four thousand discovered systems to this day. The detection methods are wideranging: direct imaging, microlensing and radial velocities are some of them.

One of the techniques that have proved most effective is the analysis of star flux curves. Essentially, we look at the intensity of the light we receive from a star for months or years. If we see a periodical 'dimming' of the light, this may be due to the presence of a planet transiting the star ('hiding' some of the light the star emits). Therefore, we catalog these cases as 'candidate' systems, and we seek to confirm if there actually is an exoplanet through other methods.

The purpose of this project is to create a Machine Learning algorithm that will predict, through the analysis of the observed flux for a specific star, whether it has a planet on orbit.

I used data collected from the NASA Kepler space telescope, throughout its different missions. The data can be found in the following link: https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data

The data was already split into training and test sets: the former has been used to create the model and the latter, exclusively to test its performance. Each row in the dataset represents an observation for a specific star. The first column is either a 1 (if there is no planet on orbit) or a 2 (if there is at least one confirmed exoplanet transiting the star). The rest of the columns indicate flux values over time.

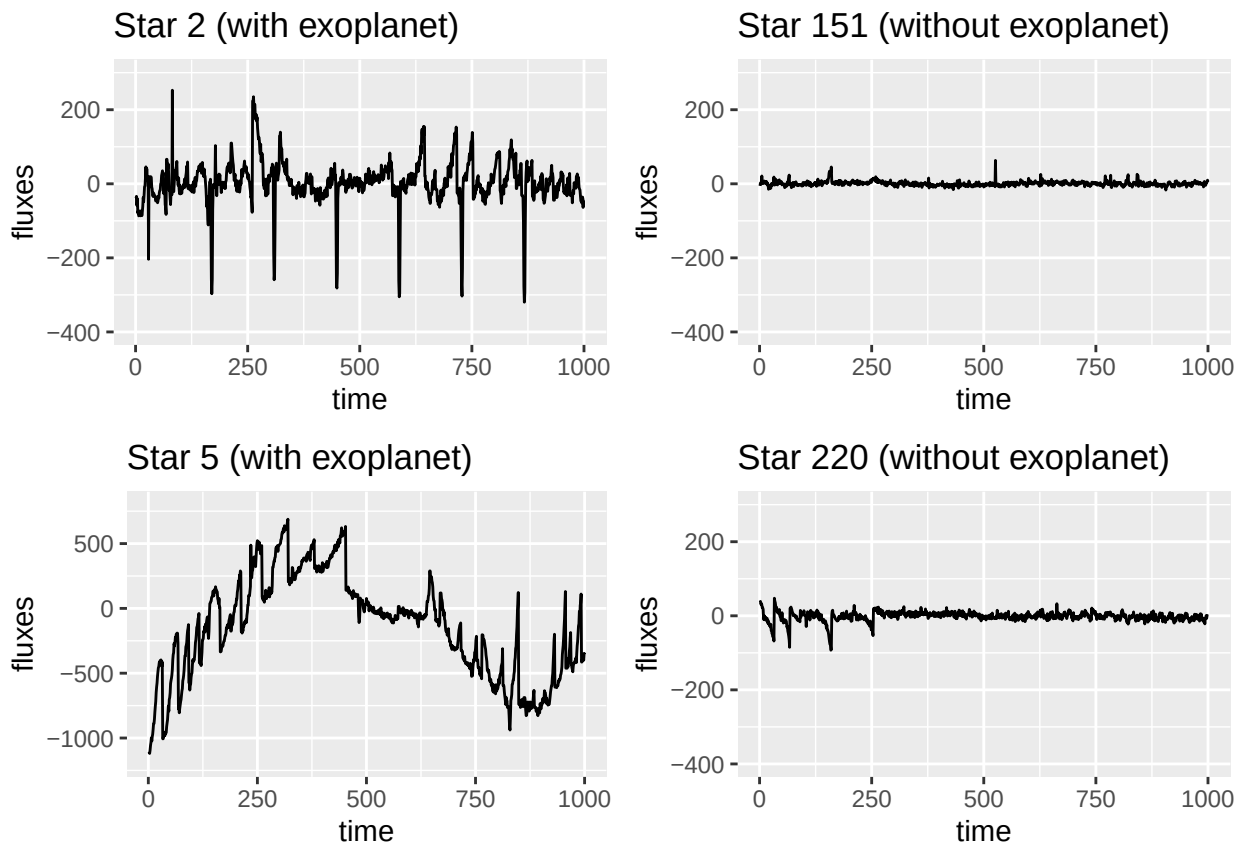
Methods and Analysis

Data exploration

To start with, we load the required libraries: tidyverse, dplyr, readr, caret, zoo, DMwR, stats, e1071, smotefamily, rms, gridExtra, smoother, GeneCycle and gt. Then, we load the dataset (which has already been divided into training and test sets) into R. We coerce both sets into data frames.

Both sets include flux values recorded at 3197 moments in time (we interpret that the same amount of time goes by between two consecutive flux values), which are presented in different columns for each specific star. The first column ("LABEL") has been transformed into a factor with two levels: a value of 1 if the star does not have a planet on orbit, and a value of 2 if it does.

Let's take a look at what the flux curves look like.



While we see a clear difference between the plots of stars with and without planets in orbit, our data is quite noisy. Plus, there are many less examples of stars with planets on orbit than cases of stars without any. Therefore, there are certain steps we must take before building our models.

Data augmentation with SMOTE

One of the main challenges that this dataset presents is that it is quite imbalanced: only 0.727% of the stars in our training set actually have an exoplanet orbiting them. We are in presence of what is called an 'Extreme imbalance' (the minority class is less than 1% of the entire set). In order to avoid our algorithm to be biased towards the majority class, it is necessary to increase the number of examples of stars with exoplanets. We have used a method called SMOTE (Synthetic Minority Over-sampling Technique) to increase the number of examples of stars with exoplanets in our training set.

What does this technique do? Instead of just replicating cases of the minority class (stars with a LABEL value of 2) like other over-sampling techniques, it creates slightly different ones. SMOTE essentially works like this: a random case from the minority class (LABEL 2 in our case) is selected, as well as its k-nearest neighbors from the same class. The new data is generated somewhere between the original example and a random k-nearest neighbor. This procedure is repeated until both the majority and minority classes have similar proportions in the data set. As a result, we get new examples which are plausible because they are close to the actual ones we got beforehand, and provide new information at the same time.

Implementing SMOTE allowed us to obtain a more balanced set to train our algorithms (with around 50% of each star LABEL). The smotefamily package has been used to achieve this.

```
### Use SMOTE Data Augment to balance the training set

## Create new training set by adding new examples of the minority class (LABEL 2)
```

```r
# Since the SMOTE function produces an object of class "gen_data", we only extract the "data" section
set.seed(1, sample.kind = "Rounding")
SMOTE_train <- smotefamily::SMOTE(train_set[,-1], train_set[,1])$data

# We extract the last column which includes the LABEL
LABEL <- as.factor(SMOTE_train[,3198])

# We add the LABEL as the first column
SMOTE_train <- cbind(LABEL, SMOTE_train[,-3198])
```

Data normalization

If we want to build a successful algorithm, it is essential that all of our observations are presented in the same scale, so that our model is not biased towards those with larger absolute values This is where normalization comes into play.

Our goal is that, across all stars, their fluxes over time are expressed in the same scale. We will be using the standard value (or z-score) for each recorded flux. What the z-score tells us is: how many standard deviations(sigma) is a certain X separated from the average value (mu)?

$$z = \frac{X - \mu}{\sigma}$$

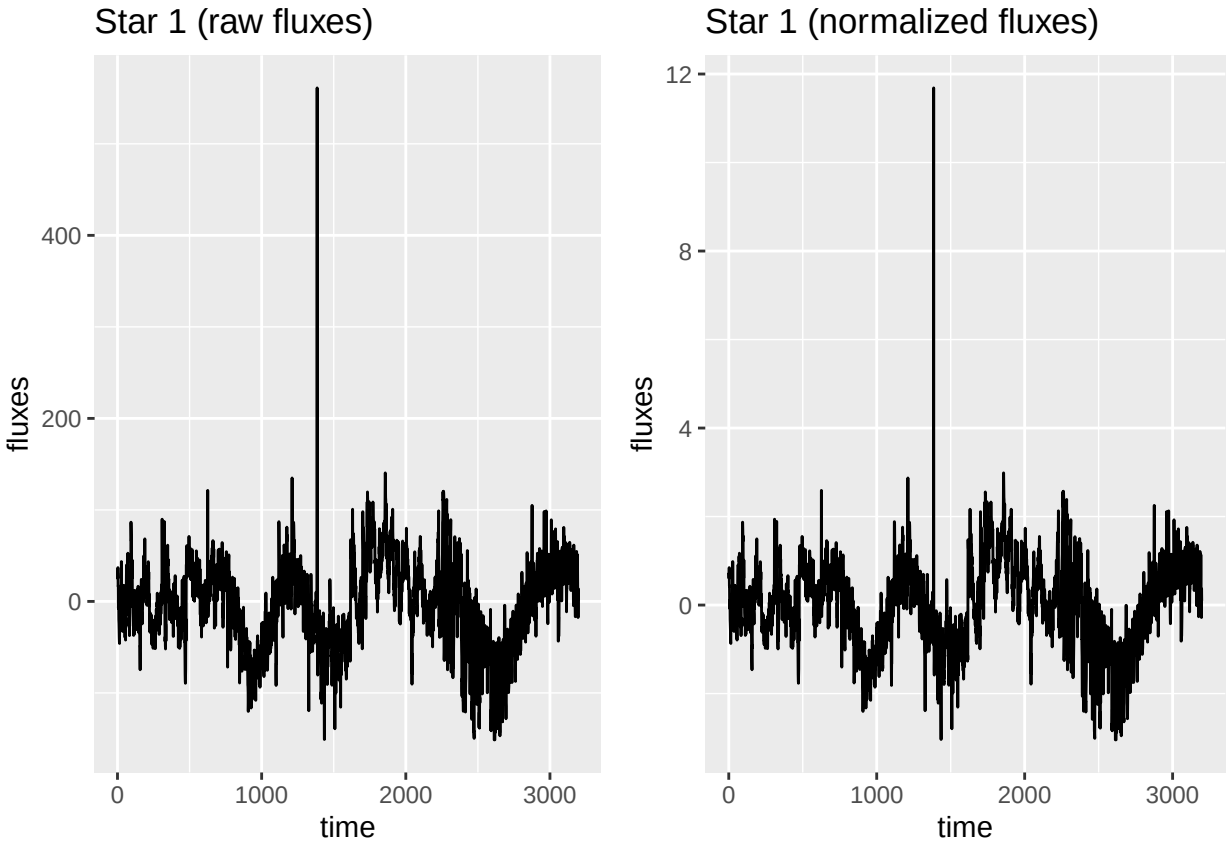
We can compute the z values using the caret package. However, I encountered a difficulty when standardizing the data: the SMOTE_train we have created has 32,242,236 entries. Therefore, I was not able to use the preProcess() and predict() functions in the caret package (my computer was not powerful enough to perform these computations). What I did instead was create a 'repeat' loop which standardized the flux values over time one star at the time. The scale() function was used to calculate the z-scores in each entry.

To reduce computation time (which for the entire SMOTE_train took almost 12 hours originally), we will select a subset of our dataset (10%) using the createDatapartition() function.

```r
### Data Normalization

## Select subset

set.seed(1, sample.kind = "Rounding")
ind <- createDataPartition(SMOTE_train$LABEL, p = 0.1)$Resample1
subset <- SMOTE_train[ind, ]
rm(ind)

## We calculate the z-score for each flux value, for each star

norm_SMOTE_train <- as.data.frame(scale(as.numeric(subset[1,-1]))) # normalize the flux values for the
index <- 1 # set initial index

# Normalize the flux values over time for each star, and add them to the existing norm_SMOTE_train
repeat {
  index <- index + 1
  if (index > nrow(subset)){break}
  #print(index)
  scaled_index <- as.data.frame(scale(as.numeric(subset[index, -1])))
  norm_SMOTE_train <- cbind(norm_SMOTE_train, scaled_index)
}
```

```r
rm(scaled_index, SMOTE_train)


# Since we now have the stars as columns, change rows for columns and add back the LABEL column
norm_SMOTE_train<- as.data.frame(t(as.matrix(norm_SMOTE_train)))
LABEL <- subset[,1]
norm_SMOTE_train <- cbind(LABEL, norm_SMOTE_train)
```

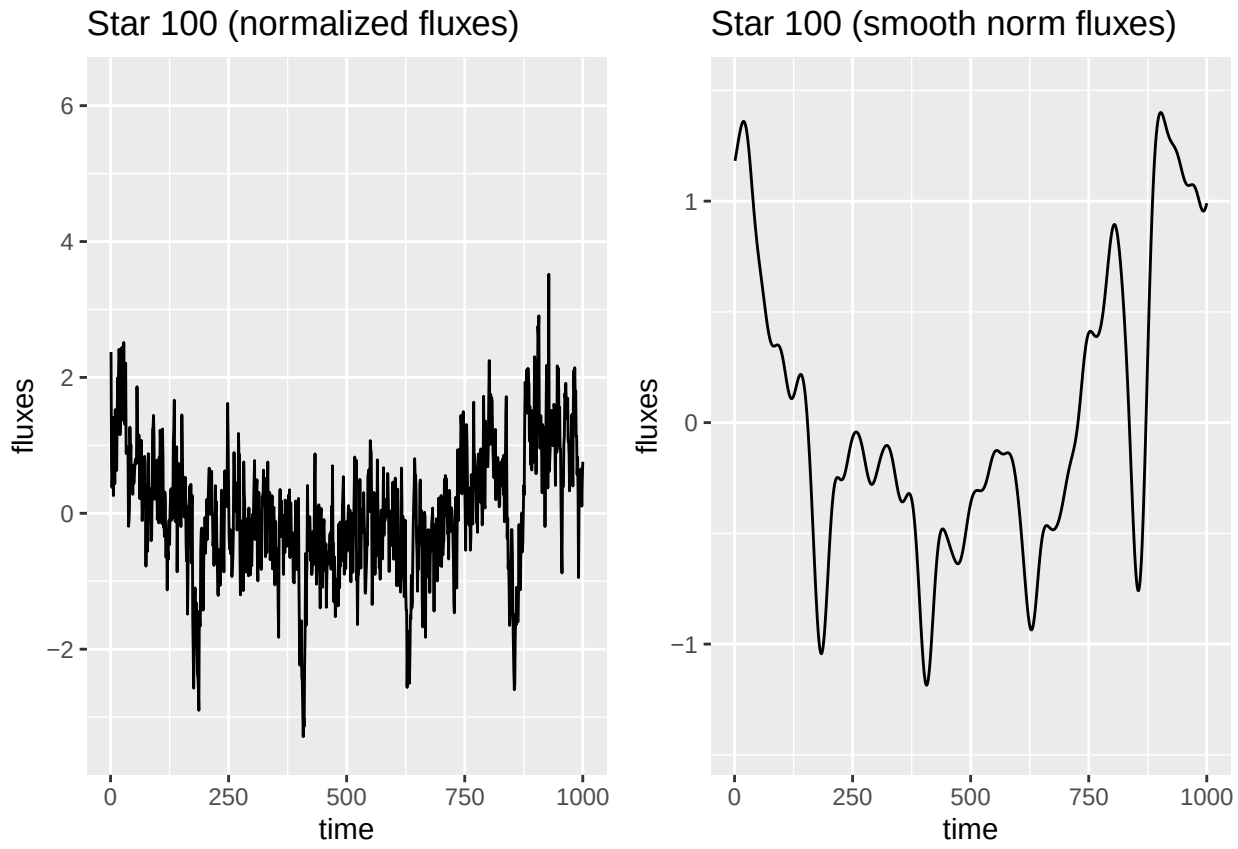
We can now see that we only changed the scale in which our fluxes are expressed.



Gaussian filter

One important part of every Machine Learning project is to make our data smoother: it is paramount to detect what exactly is noise and which data structures are actually relevant to our model. While there are many ways to carry this out (kernel smoothing, triangular moving average, among other techniques), we have chosen to implement a Gaussian filter, which is the most commonly used in exoplanet hunting.

A Gaussian filter essentially tries to approximate our data to a Gaussian function. We can see the results for star 100 in these plots:



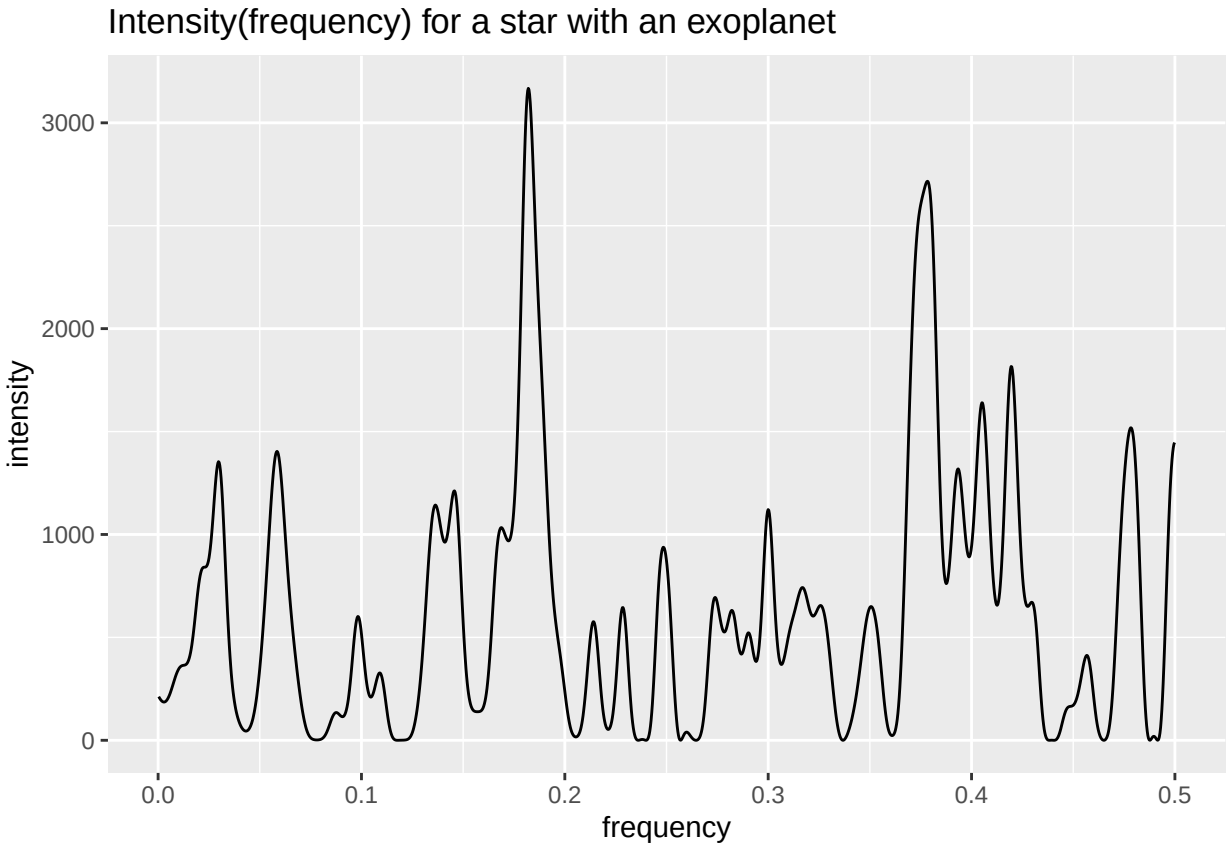
Fourier Transform

Last but not least, we will be using a method called the Fourier transform. Basically, it provides you with the opportunity to convert a time series (intensity over time) into an intensity over frequency function. The fourier-transformed function has a different structure when plotted than that of the original star, but provides us with better features for this particular classification problem. After all, our predictors so far do not give us much as far as discriminating stars with and without exoplanets is concerned (they are just flux values at a specific time).

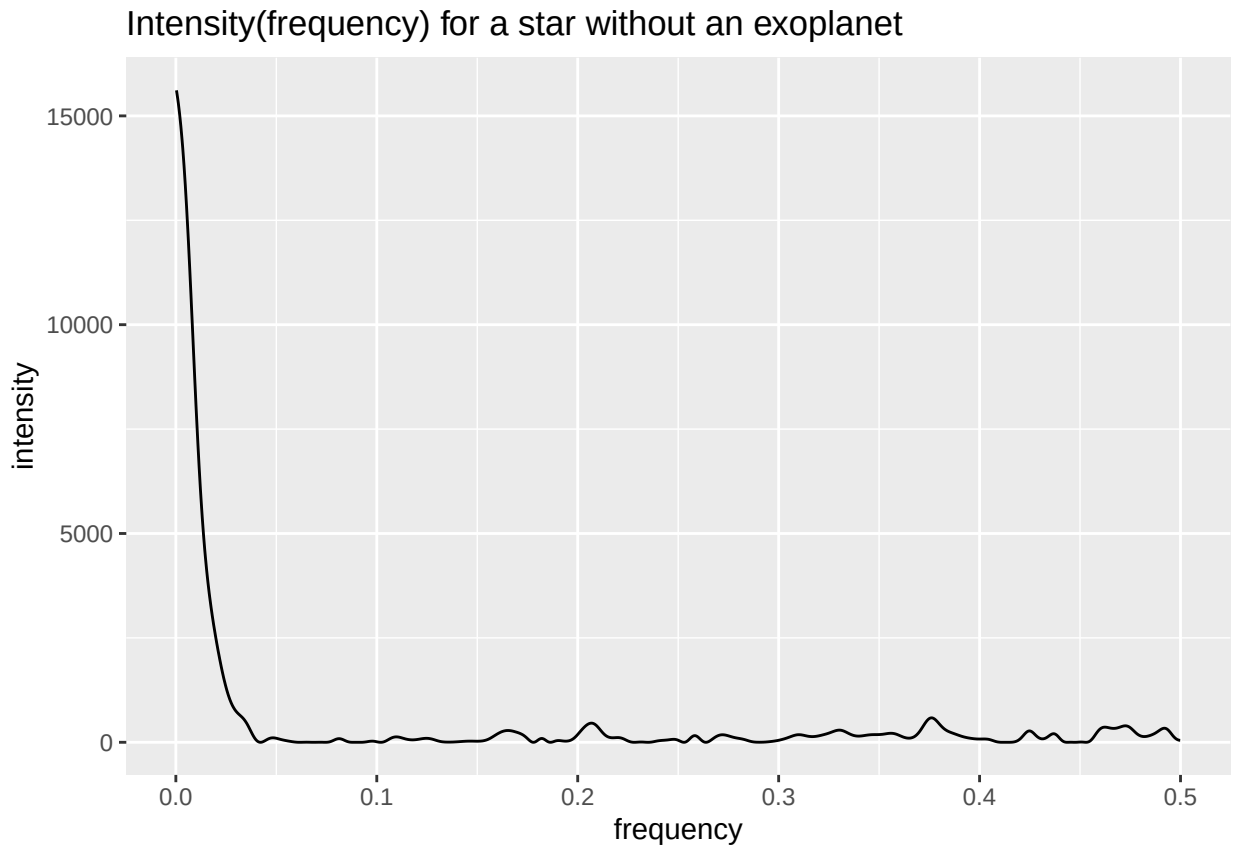
Seeing spikes at certain frequencies can provide us with a tool to determine whether a star has an exoplanet in orbit.

We have used the stats package to perform this step on our data.

Here is a plot of intensity as a function of frequency for a star with an exoplanet, obtained through the Fourier transform.



Here is an example of the same type of plot but for a star without a planet in orbit.



Success metrics

It is important to determine how we will analyze the performance of the model we are going to build. Since less than 1% of the stars in our set have a planet in orbit, we cannot use accuracy as our primary success metric (for example, if our model was to predict that no star has an exoplanet in orbit, we would get an accuracy of over 99%).

Taking this into account, we have chosen Recall as our main metric to determine the success of our model. What the 'Recall' metric will tell us after testing our model is: out of all the stars that actually have a planet in orbit, how many did we correctly predict to have a LABEL value of 2? As a secondary success metric, we will be using Precision (out of all the cases we predicted to have an exoplanet in orbit, how many actually do?). We will be taking 'Precision' into account since a model that predicts all stars to have a planet in orbit is not of much value.

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

Modeling approach

We have chosen linear regression as our basic model (to be used as a benchmark and see how we improve from its results), and then we used a Support Vector Machines algorithm (SVM). What these essentially do is determine the barrier between the different categories: exoplanet/ non-exoplanet.

The steps of normalization, smoothing and fourier-transforming have been applied to our testing set as well, to obtain the actual variables which will allow us to obtain its predicted "LABEL" values.

This is how I have fitted both models, through the lm() and the svm() functions.

```
### Fitting our models

fit_svm <- svm(LABEL ~., data = fourier_train)
pred_svm <- predict(fit_svm, fourier_test)
svm_results <- confusionMatrix(pred_svm, test_set$LABEL, positive = "2")$table


fit_lm <- lm(LABEL~., data = fourier_train)
pred_lm <- predict(fit_lm, fourier_test)

# we give those predicted values equal or above 1.5 a LABEL value of 2, and a value of 1 to those lower

pred_lm[which(pred_lm>=1.5)] <- 2
pred_lm[which(pred_lm<1.5)] <- 1
pred_lm <- as.factor(pred_lm)
lm_results <- confusionMatrix(pred_lm, test_set$LABEL, positive = "2")$table
```

Results

Regarding Recall and Precision, we obtained these results:

```
## # A tibble: 2 x 3
##   Method Recall Precision
##   <chr>   <dbl>     <dbl>
## 1 lm        0.6    0.0270
## 2 svm       0.2    0.0172
```

As can be seen, we achieve a reasonably good value for Recall in our linear regression model (correctly predicting 3 out of 5 stars with planets) while the SVM model does poorly (predicting just 1 out of 5 correctly). However, both have quite low precision (2.7% in the first model and 1.7% in the second one): both models are predicting many more stars to have a LABEL value of 2 than they should.

Linear regression appears to be more flexible in the case we are working with. This may be due to the fact that we used an artificial training set that had about the same proportion of examples for each class, which may have caused both models to be biased towards LABEL values equal to 2. However, when I tried to perform the same steps without using SMOTE, neither of the models were able to predict a single star with an exoplanet correctly. Perhaps a smaller proportion of the minority class would have guided us to better results.

Conclusions

I find it fascinating that it is possible, by simply processing the light intensity over time of faraway stars, to determine whether it has or not a planet orbiting it. While this method has been very productive, we should take into account that only some transits are observable from the Earth: if we watch that periodic flux decrease it is because the planet's orbit permits it. It depends on its inclination, since if the planet does not apparently go through the star's disk from our point of view, we will not be able to detect it through this method.

We should take into account that different methods for exo-planetary detection are implemented to confirm the results we get from Machine Learning approaches. Therefore, there is some value in the work that has been presented in this report in spite of the low Recall and Precision results: while we may miss some planets when presenting these models with new data, and we may predict some stars to have planets incorrectly, it is possible to arrive at a conclusion when using other methods (for example, radial velocities).

While some progress has been made, we might be able to obtain better and more accurate results if these computations are performed in a stronger computer, and we are not limited to a certain subset of our training set. There is a considerable amount of information we have not used and would have helped us to better train our models.

While SVM is thought to be one of the most effective algorithms when detecting exoplanets, there may be another out there that is more flexible and provides better results. In our process, it did not prove as effective as we would have expected. In the future, a training set could be generated using SMOTE that has a smaller proportion of LABEL 2 stars but at the same time not as small as in the original data. Different smoothing parameters could be tested to see if it is possible, through the same models that have been used, to achieve better results.