Draft version December 17, 2020
Preprint typeset using LaTeX style emulateapj v. 12/16/11

DEALING WITH IMBALANCED EXOPLANET DATA

Ø. Strand, I. Rivå

Draft version December 17, 2020

ABSTRACT

We consider decision trees, random forests and support vector machines on the classification of exo-planets around stars. We look at the Kepler labelled time series data from Kaggle, the goal being to correctly classify as many exo-planet stars as possible, which can be seen through the recall metric. The data is heavily imbalanced with only ∼0.73% of the training data and ∼0.88% of the test data consisting of positive samples. The data consists of a total of 5087 samples in the training set, and 570 samples in the test set.We examine the data and study how oversampling may help models predict positive samples better, by giving them more weight. Using scikit Learn's DecisionTreeClassifier and RandomForestClassifier as well as their SVM models, we make a critical evaluation of the models on both oversampled and imbalanced data. We found that sklearn's DecisionTreeClassifier on oversampled data had the best performance, with 0.5 ± 0.1 recall. SVM had a recall score on the training set of 0.39 ± 0.05 and 0.72 ± 0.18 for the testing set with 5 total instances of the minority class. The accuracy for test data was ~ 0.88 for SVM. However the parameter tuning performed was insufficient, as it did not account for variation. In this case, cross validation should have been employed to take this into account. With proper parameter tuning, we believe we would find the random forest to be the best classifier.

Subject headings: imbalanced data, SVM, random forest, decision tree, SMOTE, oversampling

1. INTRODUCTION

When we have a lot of data to go through it is quite handy to be able to filter out some of the observations that we are not interested in examining further, which is often the case with imbalanced datasets. Imbalanced datasets are commonplace in many fields like medicine, astronomy and economics. When looking for uncommon occurrences like disease, planets or fraud, one cannot expect to end up with an equal distribution. Therefore it is of the utmost importance to be aware of the issues that may arise with such datasets and be ready to deal with said issues. To get more familiar with such matters and to get a chance to examine new machine learning methods we had not used before, we decided to look at a binary classification problem concerning stars with or without exo-planets and the flux of these stars. Exo-planets around a star can be found by examining the flux over time and looking for dips in said flux. NASA's Kepler space telescope collects data from thousands of different stars, and we need a way to filter out the stars that show no promise of exoplanets in a quick and accurate way so the stars that do have exoplanets can be studied further. Looking at one star after the other manually would be tedious and time consuming, so finding a way of doing this using machine learning would be highly beneficial. Our goal was to classify as many exo-planet stars as possible, and for this reason we chose to focus on the recall metric. We concluded that miss-classifying non-exoplanet stars would not be such a big problem as long as it was not excessive, as the data could be further examined later to dispose of these stars. To reiterate, we wanted a high recall score, but we also wanted to keep the accuracy at an acceptable level. This paper will first present the dataset followed by the theory for the different methods that were implemented. It will then delve into the methodology and flesh out how the algorithms were used. The results will then be presented followed by the discussion, where we will evaluate the different models, set them up against each other and discuss the applicability of these machine learning methods. Then follows the conclusion where we will take a look at how we could have improved our analysis and summarise the discussion.

orjanstr@student.matnat.uio.no ,
ingunriv@student.matnat.uio.no
[1] Institute of Theoretical Astrophysics, University of Oslo, P.O. Box 1029 Blindern, N-0315 Oslo, Norway

2. DATA

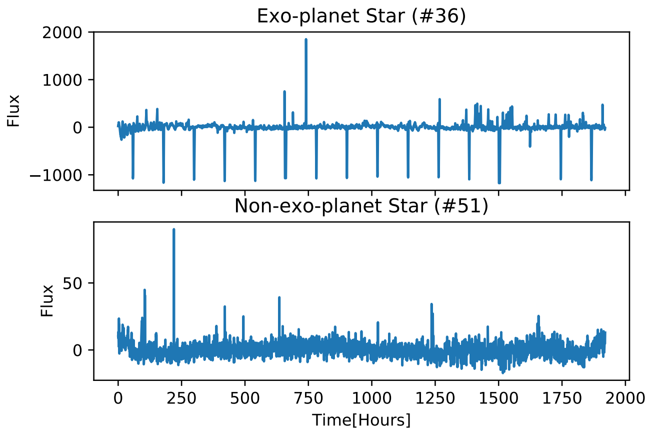

Fig. 1.— Depiction of the flux over time for two stars, the topmost plot has a confirmed exo-planet and the plot below has no confirmed exoplanets. This is the star with index 35 and 50 respectively. The flux is probably negative because of the calibration of the instruments and the way the data was gathered.

We looked at a dataset of flux measurements from stars collected using the Kepler space telescope, from kaggle [1]. The data was taken primarily from Kepler's Campaign 3, but data from confirmed exo-planet stars were added to inflate the number of positive results. The dataset consists of 3197 features, each feature representing flux for the star in question for a specific point in time. In total, the the flux was recorded over a period of 80 days. The unit of measurement for the flux is unknown, as we were not able to find information about that anywhere. The data is split into 2 files, the training set contains 5087 observations, only 37 of them being positive for exo-planets, meaning that less than 1 percent of the observations are positive. The test set had a similar distribution with 5 out of 570 observations being positive. This shows a significant imbalance in our dataset and is something that is important to take into consideration when training on the data.

The data is somewhat processed by NASA, as algorithms have been used to remove noise resulting from the telescope.

2.1. *Imbalance in datasets*

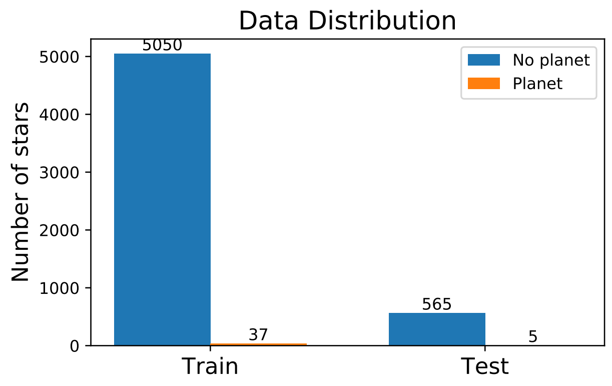

Fig. 2.— Plot showing number of instances of each class in the training and testing set.

As previously discussed, the data is heavily imbalanced. This could cause trouble as most machine learning models are built with the assumption that the data is evenly distributed. It might therefore be a good idea to somehow balance the dataset. Two ways of doing this is undersampling and oversampling. In this paper we will primarily be looking at oversampling, but we will also briefly touch on an oversampling-undersampling hybrid on one of our models (see section 4.1 for further details on this).

2.1.1. *Oversampling*

When you are dealing with an imbalanced dataset a problem that arises is that the model will be inclined to predict only the majority class. To combat this we can use oversampling or undersampling. Undersampling is reducing the amount of the majority class so the ratio between the classes is the same. Oversampling can be done in different ways. The easiest way of doing oversampling is to duplicate the minority class until you have the same amount of data in both classes. Another way of doing it is to synthesise new data from your original data. In this paper we used synthetic minority oversampling technique(SMOTE), Which selects a random point of the minority class, then it chooses a point from the n nearest neighbours of the first point. The two points in the feature space are then connected with a line and the new sample is placed on that line. This creates credible new datapoint that can be used to train our models. It is very important when we oversample that we do not split the data for validation afterwards. When we split oversampled data into training and validation sets, both sets will contain the same positive data. This is a very serious data leakage problem that is easy to oversee. If a split is not handled carefully, our models will yield amazing, yet invalid, results that in reality just indicate overfitting.

2.1.2. *Metrics*

We used various metrics to measure the performance of our model. In previous projects we have used accuracy, but as the data is imbalanced, a baseline of only negative predictions will result in $> 99\%$ accuracy. Our ultimate goal is to produce a model that can correctly classify all stars with exoplanets, and filter out most stars without planets. It is thus important to focus on the true positive rate (TP) of our model. We find that the recall is the best measure for this.

$$\frac{TP}{TP+FN} \tag{1}$$

This is a particularly useful metric, as it tells us the proportion of total exoplanets that are picked up by our model. Another useful metric is the precision.

$$\frac{TP}{TP+FP} \tag{2}$$

This gives us a measurement of the proportion of classified positives that are correct. While this is normally a useful measure, it serves little importance to us, as we are okay with classifying a moderate amount of false positives. We will, however, include it as a measure of how well our model filters out non-exoplanet stars alongside accuracy.

$$\frac{TP+TN}{TP+FP+TN+FN} \tag{3}$$

3. MODELS

3.1. *Decision Trees*

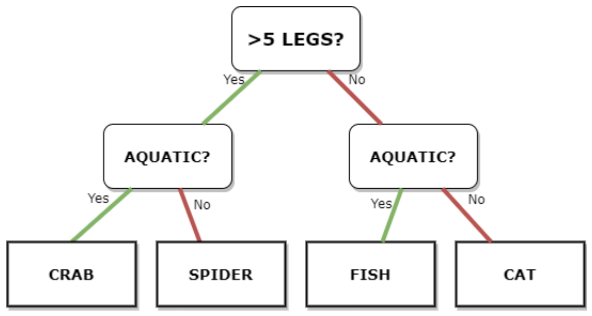

Fig. 3.— Decision tree flow chart. Made using diagrams.net [2]

Decision trees can serve a useful method in machine learning, as they provide a visual representation of how certain classifications are made. Decision trees work by filtering the data through a series of yes/no queries before eventually placing it in a class. While they can be used for both classification and regression problems, we will only be studying the classification case in this paper.

A decision tree consists of a root node, internal nodes and leaf nodes. When making a prediction, the tree will first check the requirement in the root node, and then work its way down the internal nodes before eventually ending up in a leaf node, which represents the prediction. Each node in the tree represents a feature of the data.

When setting up a decision tree, the first step is to determine which feature to place in the root node. We do this by measuring the impurity for each feature. That is, we measure the ability of a single feature to make a correct prediction. Scikit-Learn's DecisionTreeClassifier, which is the one we use, has two different ways of measuring impurity, namely with Gini and Entropy.

$$\text{(Gini) } g = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{4}$$

$$\text{(Entropy) } s = \sum_{k=1}^{K} p_{mk}\log(p_{mk}) \tag{5}$$

Here p_{mk} represents the total observations of class k in a region m (Lecture notes, week 44 [3]). We select the feature with the lowest impurity for the root node of our tree. In the example shown in figure 3, the number of legs of an animal was chosen as the root node. For the right internal node (the case of less than 5 legs) we then calculate the impurity for the rest of the features again, only this time we only look at the animals with less than 5 legs. We then do the same for the case with more than 5 legs. We again select the feature with the lowest impurity for the nodes. It is important to note that the same features can be used in both branches, as we see in the second level in figure 3. We follow this procedure until we run out of features to select from, or until the tree has reached the desired depth. The final nodes then become leaf nodes. The lower the depth of the tree, the less likely it is to overfit, meaning we can lower the max depth to introduce regularisation. If a node itself has the lowest impurity of all the features, it becomes a leaf node. In the figure, we end up with all leaf nodes in the bottom layer, but in reality we can have leaf nodes in all layers.

How do we determine thresholds when dealing with numeric values? In our data we are working with numeric flux values. In order to set the thresholds, we first sort all the values in each feature. We then take the average flux between all adjacent values. That average is where we make our split. For every average, we calculate the impurity. We then select the threshold that results in the lowest impurity. We repeat this for every feature and follow the same procedure as before. In the example shown in figure 3, using 5 as the threshold for amount of legs resulted in the lowest impurity.

Decision trees are great models to study as they are intuitive and easy to understand. Most of the parameters that we tune in this paper, serve a clear function in the model (see section 4.1.1 for more details).

3.2. *Random Forest*

The random forest classifier is an ensamble learning method comprised of a number of different decision trees. This method can be used for both regression and classification tasks. A random forest consists of a set of smaller, randomly generated trees. The result is decided according to the majority "vote" from the individual trees just like if you used decision trees with bagging. Doing it this way we will end up with a more accurate prediction with reduced variance. Due to their specific nature, decision trees tend to have high variance, meaning they are effective at predicting the same set, but are sensitive to a change in data. The important difference that enables us to achieve a good results with random forest is that the trees in random forest are not very correlated with each-other, not producing the same errors which would result in a false prediction.

When setting up a random forest, we start off by bootstrap resampling our dataset. We then build a tree as usual from the bootstrapped samples. This time, however, we only select a random set of features as candidates for each node, making it so that our trees do not always use the same features in each step to classify the data. We can then run our data through the forest, as discussed, to (hopefully) produce a more accurate result than a single tree would. A practical feature of random forests is that we can increase the amount of trees in the forest as much as we like, effectively reducing the variance without increasing the bias. The only downside is computational time. It goes without saying that using an extremely large forest would be slow. Where other models might be slow to train, predicting is also relatively slow in random forest since you have to go through all the trees to get your prediction.

3.2.1. *out-of-bag(OOB) error*

A consequence of the bootstrapping performed when setting up the forest is that we are left with many samples that never made it into the bag. We call these samples out-of-bag (OOB) samples, and they serve a useful purpose. Since they have not had any influence over the forest, we can use them as a validation set to determine the accuracy of our model. A problem with this is that we might see data leakage, where information from the training set leaks over to the test set. Like for cross validation, this method becomes useless when oversampling the training data beforehand as the training and test data will contain many of the same minority samples.

While this technique is an interesting way to evaluate the performance of our model, we will not be studying it in this paper. This is because we mainly work on oversampled data (see section 2.1.1), meaning we would see serious data leakage from the sampled data to the OOB, rendering our result misleading and useless.

3.3. *Support Vector Machines*

Support vector machines(SVM) are machine learning methods that can be used for both regression and classification problems. The SVM tries to classify the datapoints by finding a hyperplane in a p-dimensional space, where p is the number of features in our model. The hyperplane is also called an affine subspace of dimensions p-1. SVM works well on small to medium datasets as it is quite computationally demanding. The general logic in this section is taken from M. Jensen's lecture notes [4].

We have n = number of datapoints pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ where x_i is a point in the feature space, $x_i \in \mathbb{R}^{\text{l}}$, and y_i is the corresponding target. A hyperplane is defined as

$$\{x : f(x) = b + \mathbf{x}^T\mathbf{w}\} \tag{6}$$

where b is a scalar.

We want to classify our data by splitting the feature space with the help of a hyperplane.

In our case we are going to constrain ourselves to the binary classification problem, $y_i \in \{-1, 1\}$.

If we have two arbitrary points, x_1 and x_2, on our hyperplane the distance between them is $(\mathbf{x}_1 - \mathbf{x}_2)^T\mathbf{w} = 0$. The vector that is normal to the hyperplane, $\mathbf{w}^*$, is then defined as

$$\mathbf{w}^* = \frac{\mathbf{w}}{||\mathbf{w}||} \tag{7}$$

We redefine $\mathbf{w} = \mathbf{w}^*$. We then want to calculate the signed distance between our hyperplane and any point x'. We similarly, using equation 6 and 7 we get

$$\delta = (\mathbf{x}'^T - \mathbf{x}_0^T)\mathbf{w} = \frac{1}{||\mathbf{w}||}(b + \mathbf{x}'^T\mathbf{w}) = \frac{1}{||\mathbf{w}||}f(x)$$

Our hyperplane is when $f(x) = 0$, which means that we can classify a datapoint x_i , depending on which side of the hyperplane it falls by using

$$\delta > 0 \rightarrow y_i = 1 \tag{8}$$

$$\delta < 0 \rightarrow y_i = -1 \tag{9}$$

Assuming our datapoints are separable we can construct many different hyperplanes that accomplish this task, so we want to find the hyperplane with the largest margin M e.g. the largest distance between the hyperplane and the closest point defined by

$$y_i\delta = y_i\frac{1}{||\mathbf{w}||}(b + \mathbf{x}'^T\mathbf{w}) \geq M, \forall i.$$

By scaling $M = \frac{1}{||\mathbf{w}||}$, we get

$$y_i f(x) = y_i(b + \mathbf{x}'^T\mathbf{w}) \geq 1, \forall i \tag{10}$$

Maximising our margin M means that we want to minimise $||\mathbf{w}||$

$$\begin{gathered}\min \|\mathbf{w}\| \\ \text{s.t. } \mathrm{y}_i f(x) = y_i(b + \mathbf{x}'^T\mathbf{w}) \geq 1, \forall,\end{gathered}$$

This convex minimisation problem can be solved using the Lagrangian. See section 4.5.2(Hastie et al.)[5]

In some cases we are going to find that the classes are not separable by such a boundary in the feature space. In that case we have two options. The first one being a "soft margin" that allows for miss-classification of some of our datapoints. To do this we use "slack-variables" $\xi = (\xi_1, \xi_2, ..., \xi_n)$, resulting in:

$$y_i(\mathbf{x}_i^T\mathbf{w} + b) \geq M(1 - \xi_i)$$

$\forall i, \xi_i \geq 0, \sum_{i=1}^{n} \xi_i \leq$constant. From section 12.6 (Hastie et al.)[5]

By following the same logic as earlier we can write

$$\min||\mathbf{w}|| \text{ subject to} \begin{cases} y_i(\mathbf{x}_i^T\mathbf{w} + b) \geq 1 - \xi_i \; \forall i \\ \xi_i \geq 0, \sum \xi \leq C \end{cases}$$

By bounding the sum by a value C we bound the total amount of miss-classification we want to allow.

The second approach is using a "kernel function" to apply an implicit mapping of the data to a higher dimensional space where it is separable, meaning that the coordinates are never actually computed but rather the inner product in the feature space is computed, as this reduces the computational cost significantly. There are several different kernels that can be used for this transformation that work well on different data.

There is also another parameter we need to take into consideration when tuning the SVM, γ . γ is a parameter of a non linear kernel and is a measure of the influence of training samples based on their distance, or how linear the decision boundary is. The decision boundary being the visible intersection between the hyperplane in the higher dimensional space and the original feature space. A higher value means that the decision boundary is more linear. A large γ value leads to high variance and an overfit model.

4. METHODOLOGY

4.1. *Data Processing*

The first thing we did was to take a look at our data by plotting the outliers, the mean, the standard deviation and the shape of the flux in our data. After looking at the plot of the outliers,figure 4 we decided to remove the outliers. These values are probably the result of either solar flares or some malfunction in the instruments as we would not normally expect to see such spikes in flux. We are looking for dips in flux as this is something that would indicate a planet, therefore it is safe to remove these values without any fear of losing data.

We also checked for missing values, but there were none in the dataset. Since the data came pre-split, in two files, one for training and one for test, we did not split it ourselves, we only shuffled the dataset to distribute the minority class throughout the data. We also standardised out data using sci-kit Learn's standard scaler method to make sure all the features contribute equally.

Since we are dealing with a severely skewed dataset, and the instances of the minority class in the training set is only 37, we decided not to use under-sampling as it would leave us with a very small training set, and instead chose to perform oversampling. To do this we used SMOTE from Imbalanced learning and ended up with a total of 10100 datapoints in our training set. This took a long time to run with SVM, so we also made a training set with partially under-sampled data by picking 400 random values from the negative class and then oversampling the minority class to equally many datapoints so we ended up with 800 datapoints total. The labels of the dataset were also converted to one-hot encoding so the classes were 0 and 1 instead of 1 and 2 as they were originally.

We did not oversample the test set as this needs to be realistic to give a good estimate of our models' performance. It is important to not oversample the test data, but in the training set we want to prevent our model from only making negative predictions, so this makes a lot of sense.

To be sure that the dimensions of the different training and test sets were correct we made a test function that checks for the correct shape of the different arrays. Scikit Learn's methods are well tested and we did not feel it was necessary to make any test functions for them.

4.1.1. *Tuning Decision tree parameters*

We chose to use Scikit Learn's [6] DecisionTreeClassifier. We first studied how the model performed with the default parameters on our data before oversampling. We evaluated our model on both training and test data in order to get an understanding of the overfitting of our model. We ran our models several times to check for variance. We then tried oversampling our data as discussed in section 2.1.1, to study how this affects our model. Based on the results, we selected the best type of data; oversampled or imbalanced. We then ran through a select set of parameters in order to find the best set using random search. Again fitting on the training data and predicting on the test data. We adjusted the parameters `max_depth`, `criterion`, `min_samples_split` and `min_samples_leaf`.

Since the sensitivity of a tree is heavily correlated with its depth, this becomes an important parameter to adjust. As we discussed in section 2, we have over 3000 features in our dataset which, without depth adjustment, would result in a very deep tree. So adjusting this parameter, could provide us with some much needed regularisation.

The parameter, `criterion`, determines which impurity measurement to use. The two options are the aforementioned Gini and Entropy impurities (section **??**). There is no clear answer on which measurement is better, they both provide different information gain depending on the model. We thus included this in our parameter search to simply select the one that gave us the best result.

`min_samples_split` determines the minimum number of samples a node must have to split into more nodes. If a node contains fewer samples than this minimum, it will become a leaf node. This is, similarly to `max_depth`, a way to adjust the depth of the tree. The higher the minimum, the shallower the tree. Meaning increasing this parameter can introduce more regularisation.

`min_samples_leaf` determines the minimum number of samples required in a leaf node. Increasing this parameter will ensure that there are no leaf nodes that are specific to single samples, thus reducing overfitting.

There are of course many more parameters to choose from, but we chose only these four to get a more clear understanding of our models, as tuning too many parameters quickly gets confusing. We perform a random search on these parameters. We chose our parameters from the following ranges:

- `max_depth`: [2,10]
- `criterion`: Gini, Entropy
- `min_samples_split`: [2,10]
- `min_samples_leaf`: [2,10]

4.1.2. *Tuning random forest parameters*

We followed a similar procedure to the tuning of the decision tree. The goal being to reduce variance and overfitting from the decision tree model. We again studied the baseline first by using scikit Learn's RandomForestClassifier with default parameters on whichever data the decision tree performed best on. We then compared the results of the two methods. We then tried to tune our forest similarly to the decision tree. This time an important parameter is the number of trees. As discussed in section 3.2, the only effect of this parameter is reducing the variance. We try to produce many trees using the optimal parameters found in the decision tree analysis to see if we are able to reduce the variance of our model. We chose the same parameter ranges as for decision trees, but with the addition of `n_estimators` $\in$ [1,100].

4.1.3. *Tuning SVM*

The first thing we did to tune SVM was to perform a grid search with different C, γ and kernel values to see which kernel did the best job of separating the data. We found this to be sigmoid. We then kept the C values constant and changed γ to try to decrease the bias. After going through a wide range of values we picked the best range. After that we started varying C to find the model that performed the best. We checked the confusion matrix for both training data and testing data to make sure that the model was not overfit.

5. RESULTS

We will now present the results of our analysis. All oversampling has been done on training data only.

5.1. *Data visualisation*

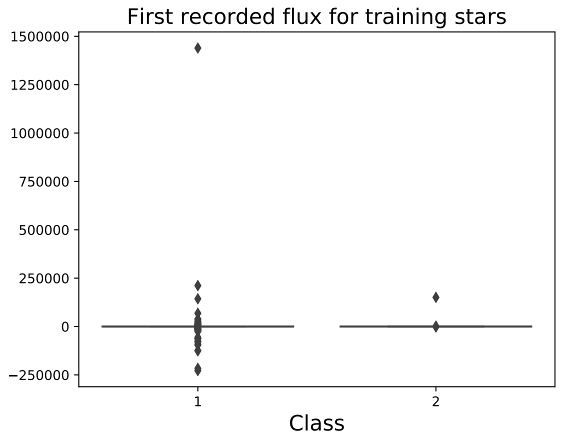

FIG. 4.— The plot shows the flux for the first recording with t=0 for all the stars.

We can see from figure 4 that we have one outlier in the majority class in the upper ranges of the flux.

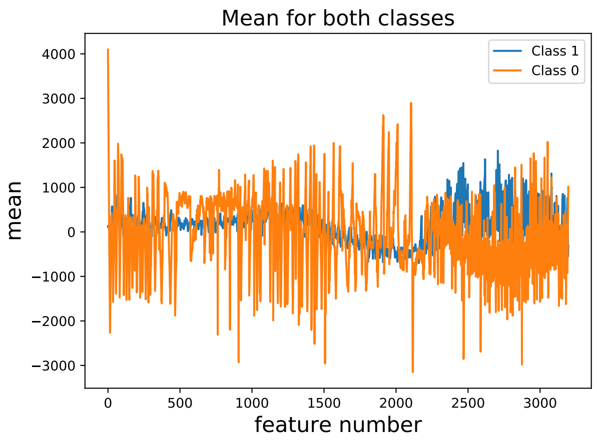

Fig. 5.— The mean of the features for class 0 and class 1.

From figure 5 we can see that the mean for class 0 varies a lot depending on the observation, and that the mean in class 1 is a bit less varied. The mean of the two classes overlap a lot.

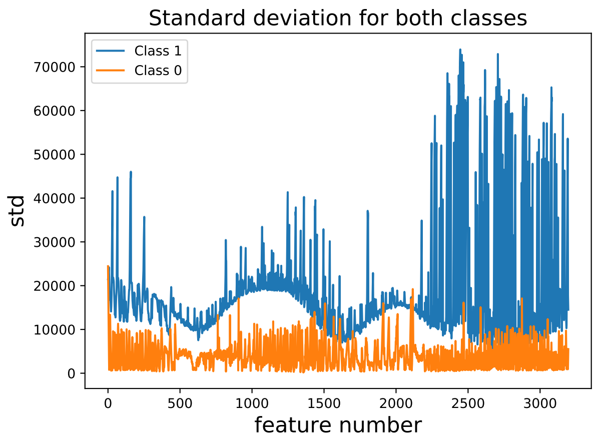

Fig. 6.— Standard deviation of the features for class 0 and class 1.

From figure 6 we can see that the standard deviation for class 1 is higher than for class 0. We can also see that there is a lot more noise in the latter half of the plot for class 1.

5.2. *Decision tree*

TABLE 1
Baseline decision tree before oversampling

–	Train	Test
Accuracy	1.0	∼ 0.982
Precision	1.0	∼ 0.222
Recall	1.0	0.4 ±0.1

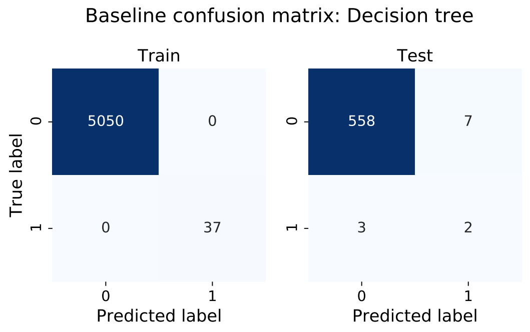

Fig. 7.— Baseline confusion matrix from sklearn's DecisionTreeClassifier. The model has not been tuned, and the prediction is made on data before oversampling. The data has been scaled.

In figure 7 and table 1 we see the results of a prediction on our data before oversampling using scikit Learn's [6] DecisionTreeClassifier with its default parameters. The model classifies all stars in the training data correctly. In the test data, the model correctly classifies 2 stars with exo-planets. The model has classified a total of 7 stars as having exo-planets. It is important to note that the variance of this model is very high. The model predicted anywhere from 0 to 4 true positives on the test data.

TABLE 2
Baseline decision tree after oversampling

–	Train	Test
Accuracy	1.0	∼ 0.970
Precision	1.0	0.167
Recall	1.0	0.5 ± 0.1

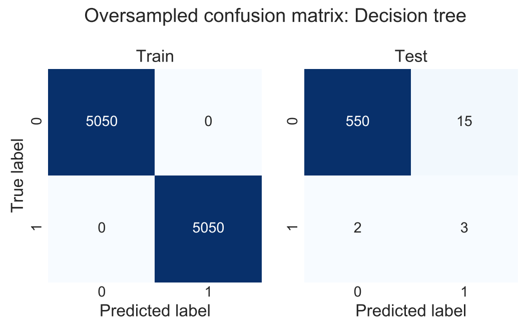

Fig. 8.— Confusion matrix of prediction made by DecisionTreeClassier on oversampled training data. Model has not been tuned, and uses default parameters

Figure 8 and table 2 show the evaluation results of the non-tuned Decision tree classifier on data oversampled using smote to achieve a 50/50 distribution. We see that the model is able to correctly classify a total of 3 stars with exo-planets. The model falsely classifies 15 stars as having exo-planets. On the training data, every star is correctly classified, resulting in an accuracy of 100%. This model consistently gets 3 or 2 true positive predictions.

TABLE 3
TUNED DECISION TREE ON OVERSAMPLED DATA

–	Train	Test
Accuracy	∼ 0.995	∼ 0.991
Precision	∼ 0.773	0.0
Recall	∼ 0.459	0.0 ± 0.0

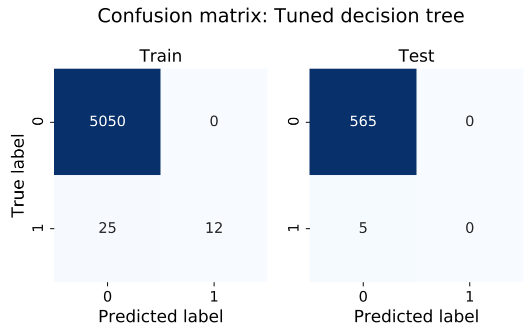

FIG. 9.— Confusion matrix of decision tree tuned using random search. Model has been trained on oversampled data.

TABLE 4
BEST DECISION TREE PARAMETERS FROM RANDOM SEARCH (25 ITERATIONS)

max_depth	criterion	min_samples_split	min_samples_leaf
8	entropy	7	3

Figure 9 and table 3 show the results of a decision tree using parameters resulting from a random search (table 4). The model achieves a recall of 12/37 =∼ 0.324 on the training data, and only makes negative predictions on the test data with no variance.

5.3. *Random forest*

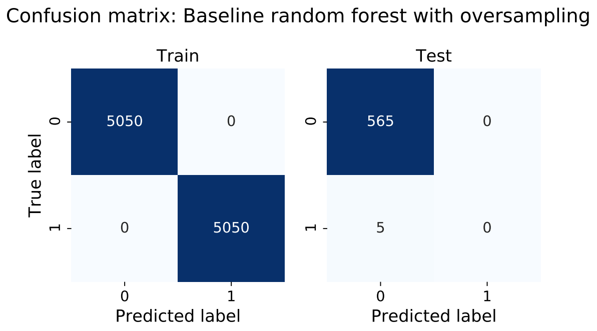

FIG. 10.— Confusion matrix of random forest using default parameters on imbalanced data.

Figure 10 and table 5 show the result of sklearn's RandomForestClassifier run on oversampled training data using default parameters. The model is 100% accurate on the training data, but only predicts negative results on the test data. These results were consistent over multiple runs.

TABLE 5
BASELINE RANDOM FOREST ON IMBALANCED DATA

–	Train	Test
Accuracy	1.0	∼ 0.991
Precision	1.0	0.0
Recall	1.0	0.0

TABLE 6
TUNED RANDOM FOREST ON IMBALANCED DATA

–	Train	Test
Accuracy	∼ 0.923	0.9
Precision	∼ 0.893	∼ 0.019
Recall	∼ 0.961	0.3 ± 0.1

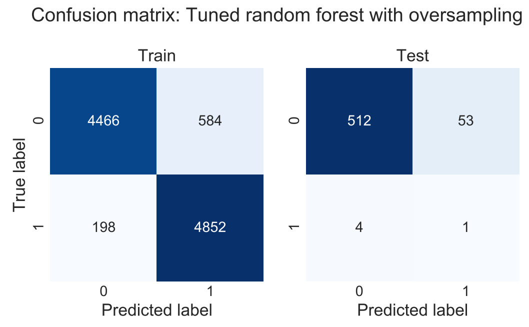

FIG. 11.— Confusion matrix of random forest on oversampled training data using parameters found in random search.

TABLE 7
BEST RANDOM FOREST PARAMETERS FROM RANDOM SEARCH (20 ITERATIONS)

n_estimators	max_depth	min_samples_split	min_samples_leaf
41	2	4	1

In figure 11 and table 6 we see the results of a RandomForestClassifier tuned using random search on oversampled training data. We see that the model has reduced overfitting on the training data, resulting in 198+584=782 false predictions. We see this particular model makes a total of 53 false negative predictions on the test data. The model generally classifies one true positive, but varies slightly, occasionally classifying two or three.

5.4. *Support Vector Machine*

TABLE 8
BASELINE SVM WITH OVERSAMPLING

–	Train	Test
Accuracy	∼ 0.53	∼ 0.99
Precision	∼ 0.99	0
Recall	0.06 ± 0.04	0 ± 0.00

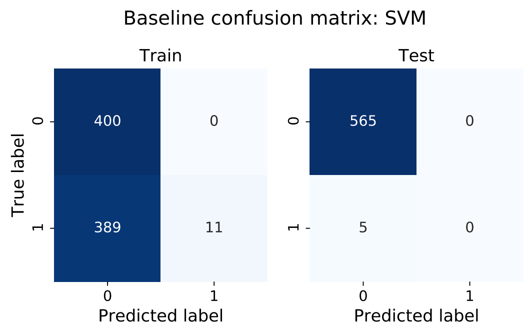

Fig. 12.— Baseline confusion matrix from sci-kit Learn's SVC. The model has not been tuned, and the prediction is made on data after under and oversampling. The data has been scaled.

With no tuning we can see from figure 12 and table 8 that almost all the predictions are false, and there are no true positives in the test data.

TABLE 9
tuned SVM with oversampling *The test data is very limited so it cannot be compared directly

–	Train	Test
Accuracy	~ 0.59	~ 0.77
Precision	$\sim$0.65	$\sim$0.03
Recall	0.39 ± 0.05	0.72 ± 0.18*

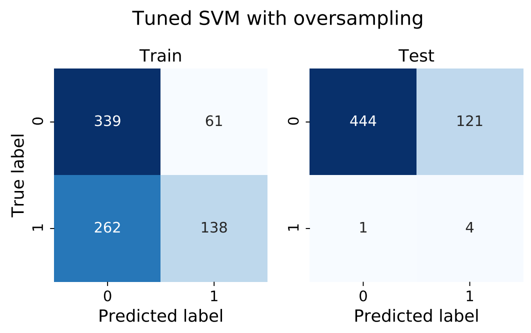

Fig. 13.— Tuned confusion matrix from sci-kit Learn's SVC. The model has been tuned, and the prediction is made on data after under and oversampling. The data has been scaled. parameters: $C = 1.5$, $\gamma = 0.36$, kernel = sigmoid.

We see from figure 13 and table 5.4 that we have some true predictions but we still miss-classify most of the positive observations. The test recall score is higher than our train recall score, but the test set has 5 datapoints only.

6. DISCUSSION

6.1. *Decision tree performance*

We can clearly see from the recall in tables 1 and 2 that oversampling has a significant effect on the variance of the decision tree model. While we still seem to have some degree of overfitting on the training data, the predictions on the test data have improved. With an imbalanced dataset, the model will be more fit to the majority class. The minority class represents such a small part of the dataset that it just gets treated as an anomaly. With oversampling, however, the minority will have much more say in the fitting of the model, meaning our model will become better at predicting positive samples at the consequence of miss-classifying some negative samples.

We can see from figure 9 that our tuned decision tree performs poorly compared to just using the default parameters. We believe this is simply due to poor parameter selection. With the way we perform random search, we do not take into account the variance of our models, as we only select the models that produce a high recall in the moment. This means that whichever model performed best in the search, could have randomly gotten a high score due to its variance, and will thus produce poorer results in further analysis.

6.2. *Random forest performance*

If we compare the baseline results from the decision tree and random forest models on oversampled data, we strangely see that the random forest has the poorest performance. Whilst the decision tree has some variance, it does at least attempt to make positive predictions, whereas the random forest only makes negative predictions. It would seem that the baseline random forest has a higher tendency to overfit. This is unexpected, seeing as the random forest is designed to reduce overfitting compared to a decision tree. We can point out that the default parameters for the DecisionTreeClassifier and RandomForestClassifier are not the same.

Perhaps the random forest uses parameters that cause the model to overfit in this case. So what happens when we change these parameters? In the tuned random forest model, we start to see somewhat reduced overfitting. The model seems to favour many shallow trees, which we assume is because we need a lot of regularisation. This results in very little variance (the model almost always predicts 1 correct exo-planet). With such shallow trees, we expect to see overly regularised results with high bias for each individual tree, but it seems that the high number of trees help the model produce a somewhat non-random result. One thing that is worth noting, is that the non-tuned decision tree seemingly performs better than the tuned random forest, producing high true positive results with little variance. We believe the cause for this is the parameter tuning done on our random forest. Since it is theoretically possible to produce a forest consisting of similar trees to the one from figure 8, it would be possible to produce similar results with a lower variance (at the cost of computational time) using a random forest.

6.3. *SVM performance*

We can see from figure 13 that the model is very biased and that these parameters do not divide the data well. With tuning we managed to get less bias, and the model does make more positive predictions. C controls the amount of false predictions we want to allow, and by changing the value we can see that this parameter indeed increases regularisation. Our model goes from being overfit to getting better predictions and a C value that is too large causes the model to underfit. We can see that a large gamma value leads to overfitting. This is because it will make the decision boundary less linear, making it able to fit more specific shapes. The variance is still quite high for the recall on the test data since classifying one positive star differently causes a big change in score, since we only have 5 positive samples. The model seems To classify a lot of the negative instances correctly but it does not seem to be able to classify the positive stars well. This might be because there is not a clear enough separation between the two classes. SVM does well when the classes are clearly separable, looking at our data we can see that the mean is quite similar across the two classes. The standard deviation has less overlap but there is some overlap especially at the end where the noise has increased. This might be a contributing factor to why we do not get the best results. Using SMOTE also creates new data and maybe some of this data can be ambiguous since the classes overlap somewhat. The pattern for a positive classification is also very varied, see appendix 8 and figure 15, this is caused by planets and suns of different sizes and different orbital periods, and might be a reason as to why is is difficult to classify.

6.4. *Critical assessment of our study and the models used*

As we mentioned in section 6.1, we did not take variation into account when performing random search. With so few positive samples in the test data, it then becomes difficult to select the best model solely based on metrics like recall or accuracy. Models with high variance can produce good results in the search, but perform poorly when tested over multiple iterations. It would therefore have been better to use something like a random search with cross validation, as this takes variance into account. However, as we discussed in section 2.1.1, this could cause misleading results as we could easily get data leakage. The best way to approach this would be to use stratified kfold cross validation to split our data into a training and validation set, and then for each iteration oversample the training set. A problem with this is that we would make the validation set highly sparse, only containing a few positive samples.

Keep in mind that both the training and the test sets are sparse. So in general, the type of data we want our model to make predictions on are sparse as well. It would perhaps then be beneficial to have a sparse validation set in our parameter search, as this would find the model best fit for the type of datasets we wish to study.

Another issue with the way we tuned our parameters, is that we tuned them to make predictions on the test set. This could mean that our model is well tuned to make predictions on the test set, but not necessarily on new data. With a validation set, we would have been able to train a model, find the best parameters, and then verify the performance with those parameters on the test set. With this form of validation, we could put a lot more trust into our model.

If we were able to perform a random or grid search with cross validation, we believe we would find that the random forest has the best performance, producing decent predictions with lower variance than the decision tree (see section 6.2 for more specifics on this).

We believe, particularly with this dataset, that we should have processed the data a lot more. For example, the dataset is very noisy, especially the seconds half of the class 0 data. While the noise from the telescope itself has been removed, there could still be a lot of noise coming from space, such as background radiation or general light pollution from other stars. Removing this noise could reveal clearer patterns that can help our models make better predictions.

Scaling based on the oversampled training set will not be the same as scaling based on the training set since the mean and std of the classes are different. We chose to scale based on the oversampled training set.

7. CONCLUSION

We learnt that very skewed datasets with few instances of the minority class can be very demanding to work with and it is not always as simple as just applying oversampling.

We found that oversampling gives more weight to the minority class. This is useful in cases where correctly classifying all positive samples is crucial, but missclassifying some negative samples has little consequence. In order to produce more reliable models, we should have tried using cross validation for our random search, as this takes variance into account when selecting the best parameters. We would then produce a model tuned towards making predictions on sparse imbalanced data, which is the type of data we want our model work on. While we found that the decision tree had the best performance, we believe that, with proper tuning using cross validation, we would eventually find that the random forest produces a much more reliable model. We also saw that SVM can be quite computationally demanding and might not work well on large datasets without undersampling.

We also learnt that the data processing step is quite demanding, and requires a lot of knowledge about the field and ways to clean the data to make the distinguishing features more clear for the models. This is an important step in many machine learning processes, as the data is often contaminated, noisy and unclean. It would be interesting to try using a convolutional neural network to see if we would be able to get better results seeing as a CNN "automatically" does some data processing in the sense that it can draw out certain unique features to the dataset that would be difficult for a human to pick up on.

It would be nice to try some other techniques for artificially creating data from the dataset. Since the orbits are periodical maybe we could get a better performance if we shifted the data and added some noise. Another thing that would be interesting to look at is different ways of processing the data, different filters, a fourier transform, or ways to do feature selection, but we did not have enough knowledge about the topic to feel confident enough about processing the data correctly. Comparing the difference between scaling based on the oversampled data and the original data would also be nice to look into.

8. APPENDIX

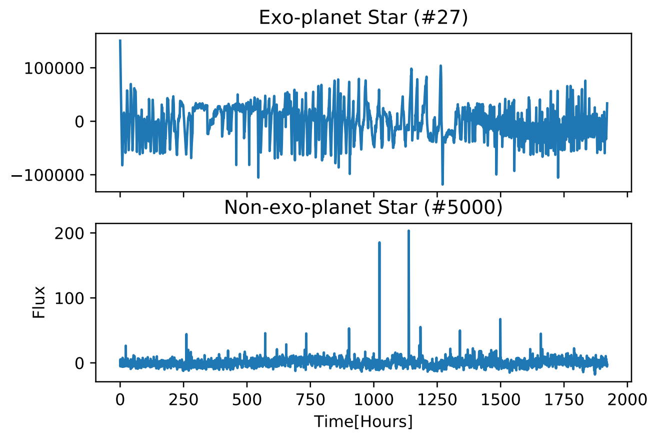

Fig. 14.— Depiction of the flux over time for two stars, the topmost plot has a confirmed exo-planet and plot below has no confirmed exoplanets. This is star with index 30 and 999 respectively.The flux is probably negative because of the calibration of the instruments and way the data was gathered

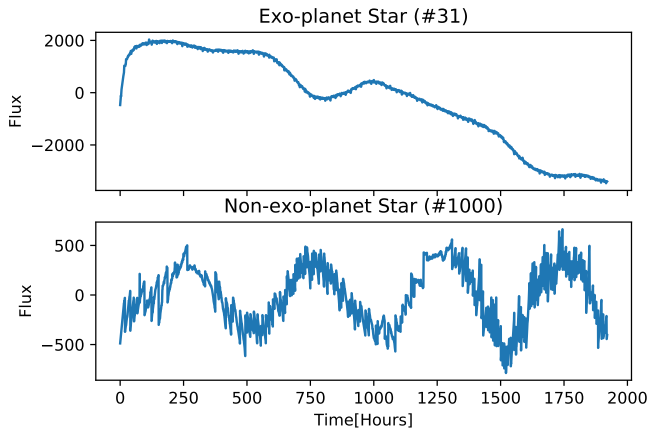

Fig. 15.— Depiction of the flux over time for two stars, the topmost plot has a confirmed exo-planet and plot below has no confirmed exoplanets. This is star with index 26 and 4999 respectively.The flux is probably negative because of the calibration of the instruments and way the data was gathered

REFERENCES

[1]Kaggle user: WΔ. Exoplanet hunting in deep space: Kepler labelled time series data. https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data, 2016.

[2]Open Source. diagrams.net. https://app.diagrams.net/.

[3]Morten Hjorth-Jensen. Week 44: From decision trees to bagging methods. https://compphysics.github.io/MachineLearning/doc/pub/week44/html/week44.html, 2020.

[4]Morten Hjorth-Jensen. Week 46: Gradient boosting summary and support vector machines. https://compphysics.github.io/MachineLearning /doc/pub/week46/html/week46.html, addendum = (accessed: 10.12.2020),, 2020.

[5]T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009.

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