ML driven Exoplanet Parameter Estimation

Andronis J. (4783), Karagianni D. (4659), Filippou D. (4726)

Department of Physics, University of Crete

1. Introduction

In recent years, there has been a rapid increase in the amount of data regarding exoplanets. However, features like Mass and Radius, which are essential in order to further expand exoplanet research, are rarely known simultaneously, and thus deriving a relationship between the two is very beneficial. This is exactly what we attempted to do in this project. We made use of data from the Open Exoplanet Catalogue, containing over 3000 exoplanets along with their various characteristics. Parameter estimation was achieved by implementing different Machine Learning regression algorithms , such as LASSO, Ridge, Elastic Net, Random Forests and SVMs, by testing their relative performance and by comparing them. Also, Randomized Search and Grid Search were used in order to optimize each model. The evaluation of each model was done by testing its cross validation performance on a test set and by analyzing other metrics like RMSE or the R^2 score.

2. Methodology

The first step in our project is to access the data and preprocess it. We did so by accessing the catalog's github repository[1] using our custom function, `fetch_OEC_data`, which downloads the most recent version of the dataset and returns a pandas DataFrame. We then use a second custom function, `prepare_data`, in order to prepare the data for the regressor algorithms, by dropping some features that do not contain useful information or that are not physically motivated. The function also calculates the Equilibrium Temperature and the Stellar Luminosity of the star of each system, to increase the number of samples in each feature. Finally, we set up another function ,`prepare_data_mass_only`, which only utilizes the Planet Mass feature.

We also implement a few "Helper Functions" . The `split` function splits the dataset to a training set and a test set, the `learning curves` function produces the learning curves of a

[1] The OEC catalog has a separate repository that contains all the data of the base catalog but in ASCII text format that is easily accessible at: "OpenExoplanetCatalogue/oec_tables: Simple ASCII ... - GitHub." https://github.com/OpenExoplanetCatalogue/oec_tables. Accessed Dec. 2020.

desired model, and finally the `evaluate` function calculates the Cross Validation score as well as the RMSE score of a model.

In the next steps we make use of different models. As mentioned above, Grid Search will be used for hyperparameter tuning for each of these models. `GridSearchCV` is a Scikit-Learn function and it receives predefined values for hyperparameters. It then tries all the combinations of the values that it has received and evaluates the model for Cross-Validation. As a result, after using this function we get accuracy and loss scores for every combination of hyperparameters and we can choose the one with the best performance. On the other hand, the `RandomizedSearchCV` function does not try out all the parameter values, but chooses parameter combinations randomly and returns the best estimator.

After training the models and doing our best to improve them in every way, we evaluate each model and decide which one provides us with the best scores and best predictions to use. The performance of the models is discussed in greater detail in sections 2.3-2.5.

2.1 Data Preprocessing

In this section we'll discuss the preprocessing of the data in greater detail. Observing our dataset we can see that it contains a lot of missing values, so we needed to drop them using the `dropna` function from the Pandas library. We can see the number of samples for each feature in Fig 1.

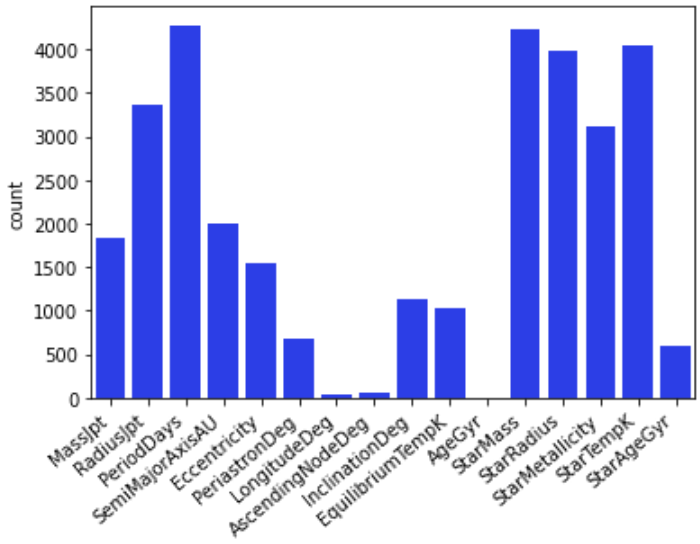


Figure 1. Number of samples for each feature that we were interested in.

Before we removed the missing values from our dataset we computed the Equilibrium Temperature of the planets with sufficient data, using the following equation (Laughlin, Lissauer, 2015, 25):

$$T_{eq} = \left(\frac{R_{star}}{2a}\right)^{\frac{1}{2}} \frac{T_{eq,\ star}}{(1-e^2)^{\frac{1}{8}}}$$

where R_{star} is the radius of the star, a is the Semi Major Axis, $T_{eq,\ star}$ is the equilibrium temperature of the star and e is the eccentricity of the orbit. Using this new data we are able to

increase the number of samples of equilibrium temperature from 1032 (at time of accessing) to 1593. To add more data for training we computed a new feature to our dataset, the Stellar Luminosity, using to the known equation:

$$L = 4\pi R^2 \sigma T_{eq}^4$$

Where R is the radius of the star and T_{eq} is the black body equilibrium temperature. Even after calculating these features the largest possible number of planets in the final dataset was 455, meaning after a 75-25 split the training set contained 341 planets, while the test set 114 planets. Lastly we trained a Random Forest Regressor (without any regularization of optimization) to identify what features were most effective in producing an accurate result. Mass was the most important with a feature importance score of 0.75, as we expected from analysing the relations between our data in Fig 2. and the relevant scientific literature.

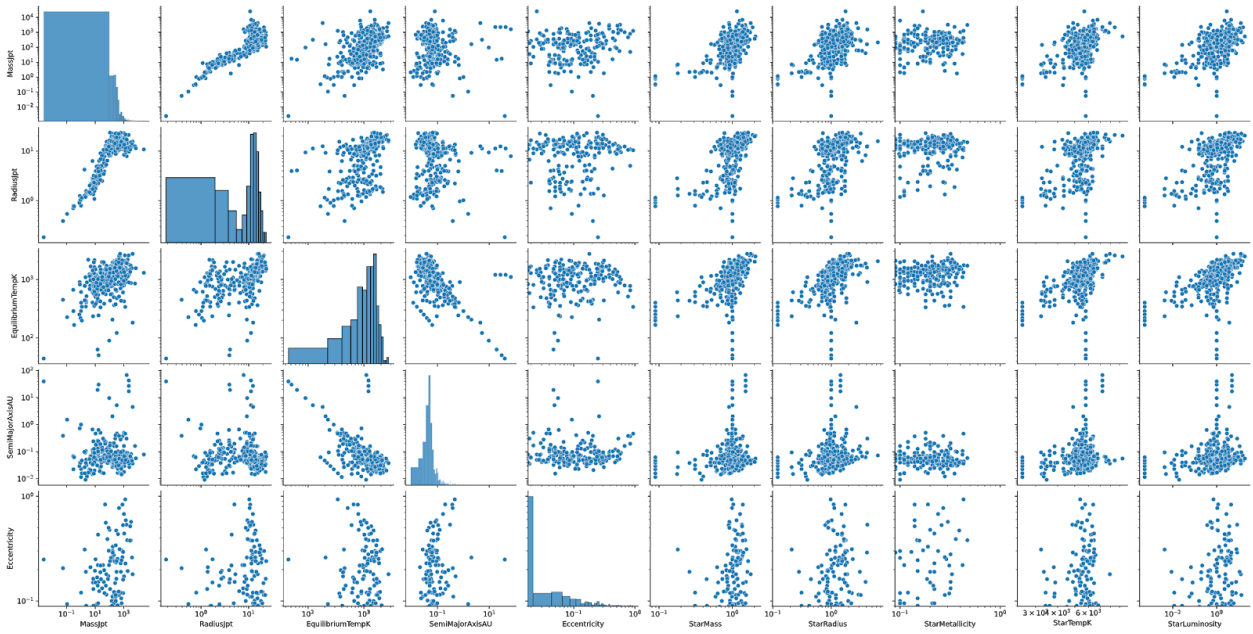


Figure 2. Pairplot of the combinations of features corresponding only to an exoplanets parameters.

As we will see in the following sections most models require all features in the preprocessed "`prepared_df`" dataframe simply because it provides more data for training. Some SVMs were the only models that performed better when they used only the mass to make a prediction of the radius.

2.2 Evaluation Methods

The evaluation process was composed by a function that computed the RMSE of our model and a function that created learning curves for our model while also returning an 8 fold cross validation score. The RMSE was computed on a validation set comprised of 25% of data from the training set. We adjusted the hyperparameters of our training according to these performance metrics.

2.3 Linear Methods

The first methods we implemented were the linear methods Ridge, Lasso and ElasticNet regressions. They are regularized versions of linear regression trying to reduce overfitting by adding a regularization term at their cost function forcing the learning algorithm to keep the model weights small. In all our models we have scaled the data using the Standard Scaler function as in regularised models this is a very important step. If skipped, features with small values may be unfairly punished.

2.3.1 Ridge Regression

Ridge Regressions' loss function is the linear least squares function with regularization given by L2 Euclidean norm ||x||= $\sqrt{(|xi|^2 + |xj|^2}$ with a hyperparameter controlling the strength of the regularization. If this hyperparameter α = 0 then Ridge Regression is just Linear Regression. If alpha is very large, then all weights end up very close to zero and the result is a flat line going through the data's mean. In our project we chose it's value between three possible options [0.01 , 0.1 , 1] using the GridSearchCV function and we ended up with the default value a=1. After fitting the model we got scores of 0.325 (RMSE) and 0.287 (mean cv score).

To get a better understanding of its accuracy we also get the following graphs. Studying the true vs predicted radius graph we can see that our model isn't able to accurately predict most of the data. Also our models lack the complexity needed to fit the problem as concluded from the learning curves reaching a plateau at 0.5 fast and being very close to each other. In conclusion the Ridge Regression model has a high bias and isn't a good choice for our specific problem.

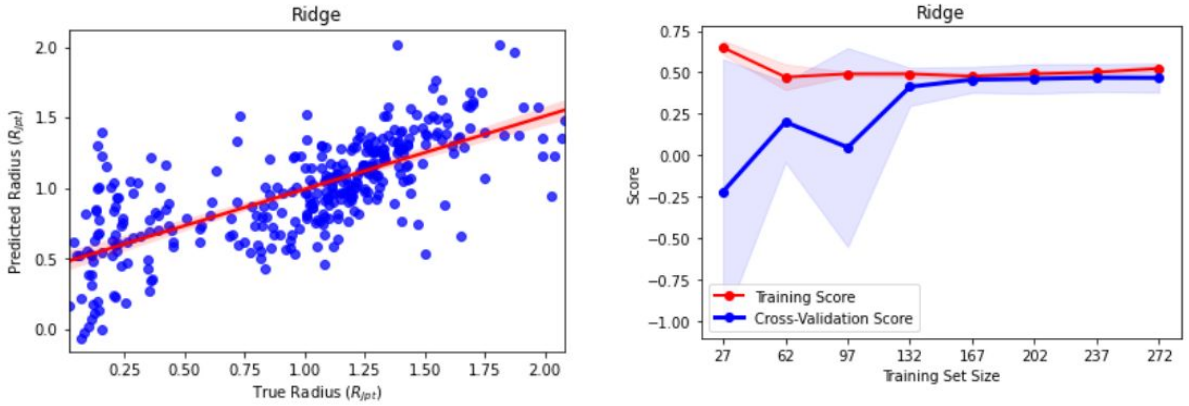


Figure 3. Plots of the predictive performance of the Ridge Regression model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.3.2 Lasso Regression

Following the same principle as Ridge regression, Lasso utilizes the least squares loss function adding a regularization term, this time using the L1 norm ||x||=|xi|+|xj|. In this norm, all the components of the vector are weighted equally. An important characteristic of Lasso

Regression is that it tends to completely eliminate the weights of the least important features. In other words, Lasso Regression automatically performs feature selection.

Getting started to fit the model we first use the GridSearchCV function to once again find the optimal value for our hyperparameter alpha which this time was set to 0.01 in contrast to the default value of 1.This change is possibly cause of the scaling we did to the data before we fed them to the algorithm. Fitting the model we got scores of 0.326 (RMSE) and 0.358 (mean cv score) and the plots below. From the first graph we observe a similar result as the Ridge Regression but with similarly high bias as seen by the learning curves reaching the plateau at 0.5 very fast. That's probably the reason we got higher errors as well, leading us to the conclusion that Lasso Regression is a bad fit for our project.

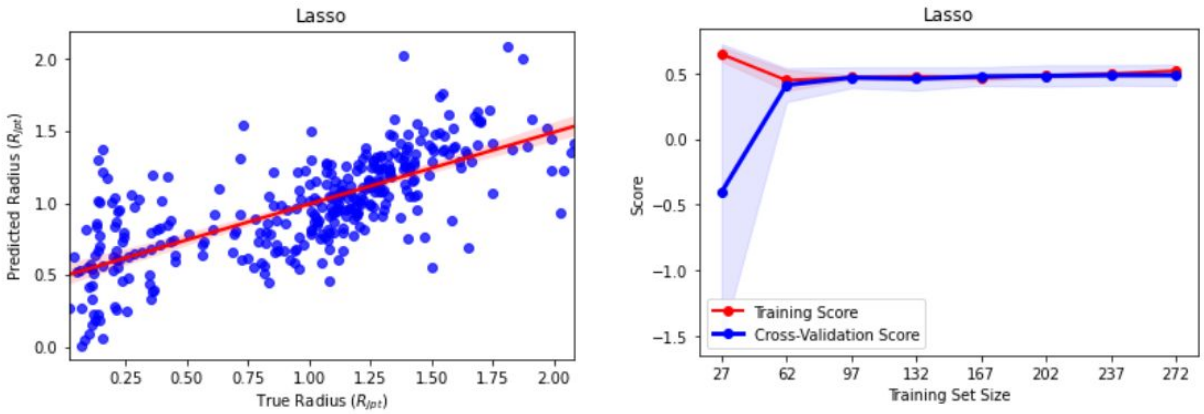


Figure 4. Plots of the predictive performance of the Lasso Regression model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.3.3 ElasticNet

The last kind of regularised linear regressor we used was ElasticNet, a middle ground between Ridge and Lasso Regression. The regularization term of ElasticNet is a simple mix of both Ridge and Lasso's regularization terms,where we can control the mix ratio r. Running a GridSearchCV for the hyperparameter alpha but also the ratio between the two models we found that the optimal was a ratio of 0.2 , a result we expected as we already concluded that the Ridge model gave us better results.

After fitting the model we get scores of 0.333 (RMSE) and 0.456 (mean cv score) and the graphs below. There is no noticeable improvement in the true vs predicted radius graph while the learning curves are again at 0.5 score being very close to each other. The conclusion is a model with high bias not suitable for our project.

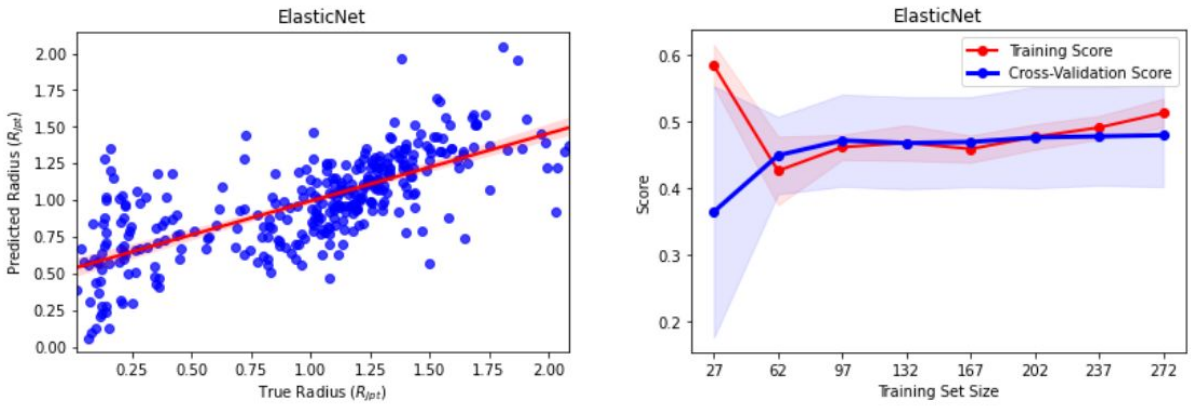


Figure 5. Plots of the predictive performance of the ElasticNet Regression model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.3.4 Polynomial Regression

The last step we took in order to find a good fit for our problem using regularised linear models was to apply the `PolynomialFeatures()` function for a second degree polynomial before fitting each of the three models mentioned before. The `GridSearchCV` function was again used for obtaining the optimal values of our models' parameters. However the models were severely overfitting the data giving extremely bad scores. We tried limiting the alpha hyperparameter to only really big values and we managed to get the scores shown in the table below.

Metrics	*Ridge*	*Lasso*	*ElasticNet*
RMSE	*0.310*	*0.388*	*0.337*
CV-score	*-1.380*	*0.346*	*0.373*

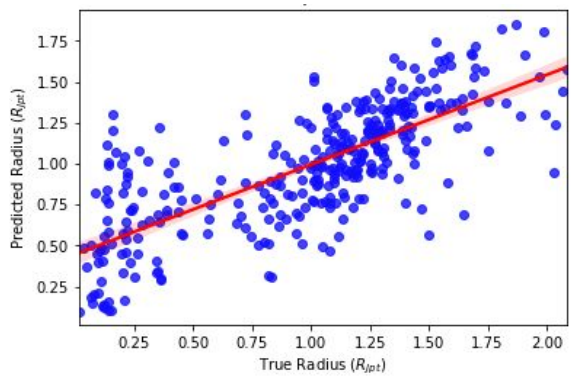
Table 1. The scores of the three regularised models when using the Polynomial Regression method

Changing the degree of the polynomial to a higher value made the overfitting problem even worse so we settled for a 2nd degree. Furthermore the usual graphs are shown below.The true vs predicted radius still shows no improvement while the learning curves show signs of overfitting . The Ridge model has a training score close to 1 while the validation is very low, a classic sign of overfitting. The Lasso model didn't give a score better than 0.3 and the Elastic Net with an optimal l1_ratio given by the GridSearchCV fitting essentially a Ridge model and an even bigger hyperparameter gave a score close to 0.4.

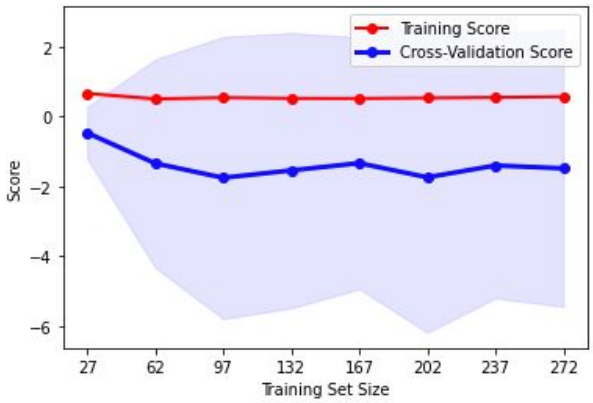
After studying all the graphs and scores acquired from the regularized linear models we conclude that none of these models is a good fit for our project. That is possibly because we wrongfully assumed the problem was linear. So we continue the search with an investigation of the SVM regressors.

Ridge

True vs Predicted Radius

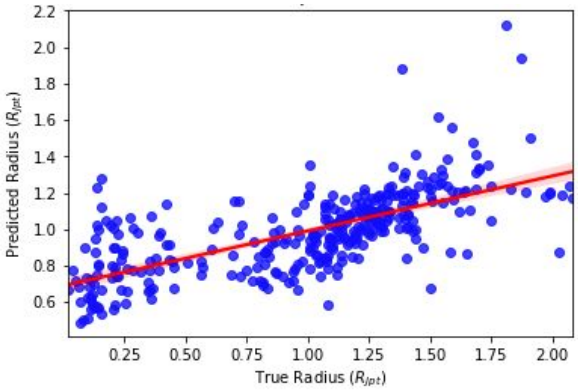


Learning Curves

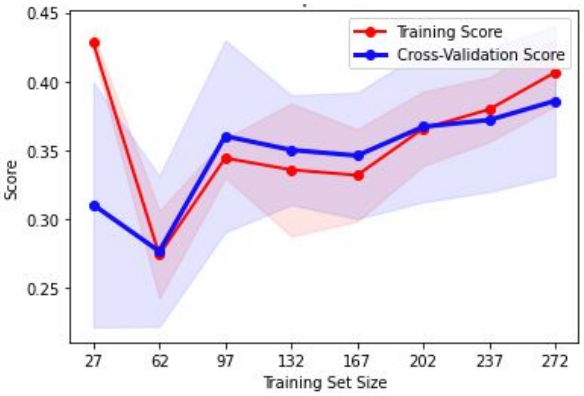


Lasso

True vs Predicted Radius

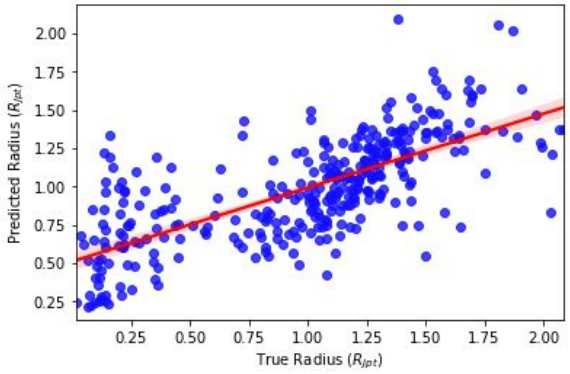


Learning Curves



ElasticNet

True vs Predicted Radius



Learning Curves

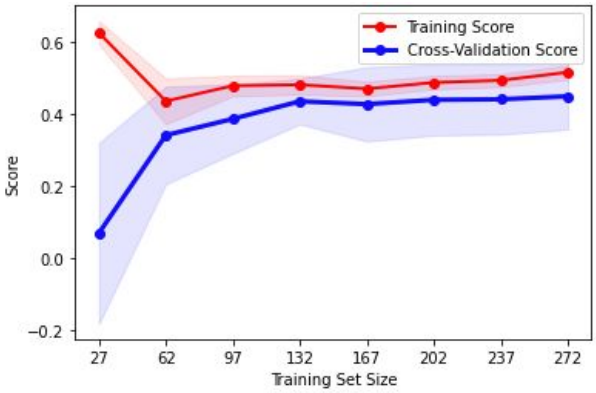


Table 2. Plots of the predictive performance of the Polynomial Regularization models. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.4 SVMs

In the second part of our project we implemented different variations of the Support Vector Machine model. SVMs are supervised models and are most commonly used for classification tasks, however they are a very reliable model for regression as well. A Support Vector Machine in general receives data and returns a hyperplane, which in 1-D it is just one line. This hyperplane(or line) is called the decision boundary and it separates the classes, when we face a classification problem.The technical difference between Support Vector Classification and Support Vector Regression is shown in Figure().

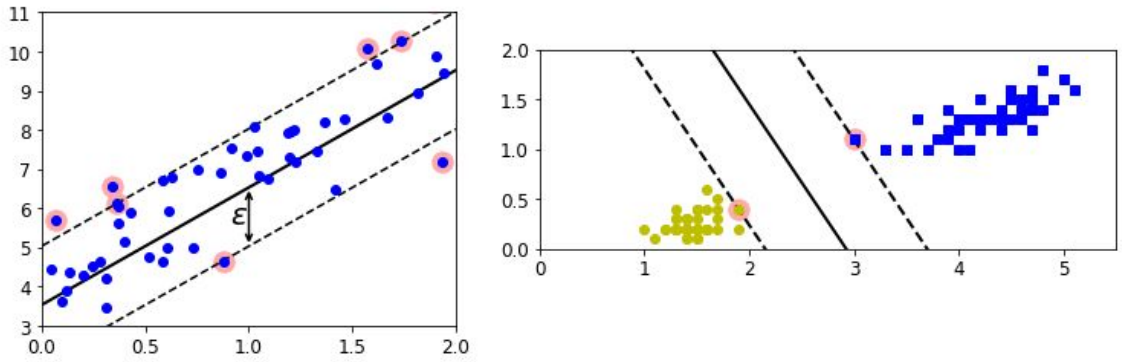


Figure 6 :Plots of the predictive performance of the Linear SVR model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

A very useful tool that SVMs use are mathematical functions called kernels. A kernel receives data as input and transforms it to another form that is easier to use. Some of the most popular kernels for the SVMs are the Linear Kernel, the Polynomial Kernel, the Gaussian RBF(Radial Basis Function) Kernel and the Sigmoid Kernel, all of which were put to use in this project.

In general, the SVMs have a few hyperparameters that we attempted to tamper with. The 'C' hyperparameter controls the amount of regularization that is applied in the model. Larger values of C might cause overfitting while smaller values might cause underfitting. On the other hand, the 'epsilon' hyperparameter expresses the width of the SVM's 'street'. The Gaussian RBF kernel also brings forth the 'gamma' hyperparameter, which determines at what extent the decision boundaries will actually surround the received data. If this hyperparameter is small, points that are far away from each other will be considered similar(and they will be grouped together), whereas if it is large, points will be closer together(overfitting may occur).

Also, please note that we have scaled our data, using the `StandardScaler` method and for some kernels we used only the Planet Mass as an input feature, due to the results of the

Feature Importance investigation. This was decided after observing that some kernelized SVMs perform better when trying to figure out the relation between multiple parameters in contrast to others which perform better with only one feature.

2.4.1 Linear SVR

The Linear Support Vector Regression is the first method that we used in this section. It is similar, but not the same, with the SVR using a linear kernel, which we examined in 2.2.5. In the first lines of the code we defined a function with the command `def linear_SVR` that takes four arguments: `dataset`, `fit`, `scores` and `plot`. We then split the dataset into subsets and perform a `GridSearchCV` to tune the hyperparameters of our model. Finally, we showed a few plots, the Predicted Radius vs. True Radius, and the learning curves of this particular model. We obtained a Root Mean Square Error score of 0.32 and a Cross Validation score of 0.40. Note that we used all features here, not just the Planet Mass. From the learning curves graph, we observed that the `LinearSVR` predictor is unable to produce satisfactory results, because it fundamentally assumes a linear fit, which is obviously not the case. Thus, we have a high bias error.

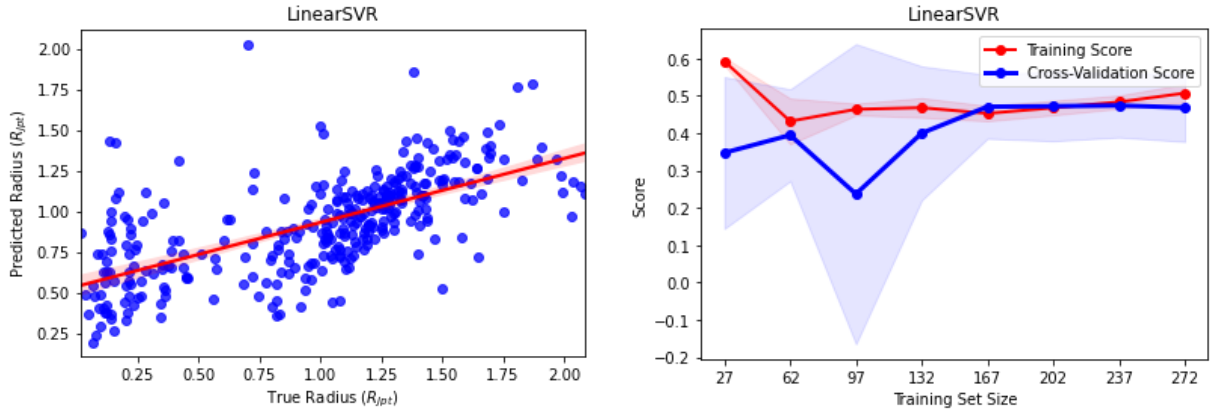


Figure 7:Plots of the predictive performance of the Linear SVR model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.4.2 Polynomial Kernel

We proceeded to try and implement a few kernels , to test whether our result will improve or not. The first kernel we will use is the Polynomial Kernel which adds polynomial features in the model. Following the same procedure as before, but this time using only Mass, we obtained an RMSE score of 0.28 and a cv score of 0.18. The graph we acquired indicates that the polynomial kernel needs more data for training, in order for us to get better results. Again, we have a high bias error, because the polynomial relationship we assumed is not representative of our data.

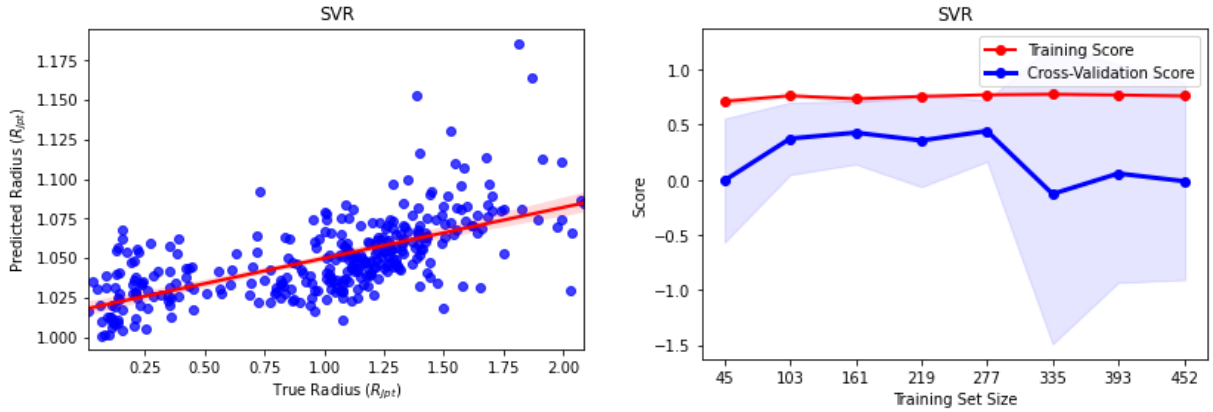


Figure 8:Plots of the predictive performance of the Polynomial Kernel model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.4.3 Gaussian RBF Kernel

The Gaussian RBF Kernel was the next in our list. This particular kernel, as mentioned before, has the extra hyperparameter 'gamma'. Thus we included it in our Grid Search. Again, using only the Mass feature, we obtain the following plots, an RMSE score of 0.31 and a cv score of 0.44. From the plot below, we observed the Training Score and the Cross-Validation score display the same ascending behaviour, but again we have insufficient data to obtain a good fit.

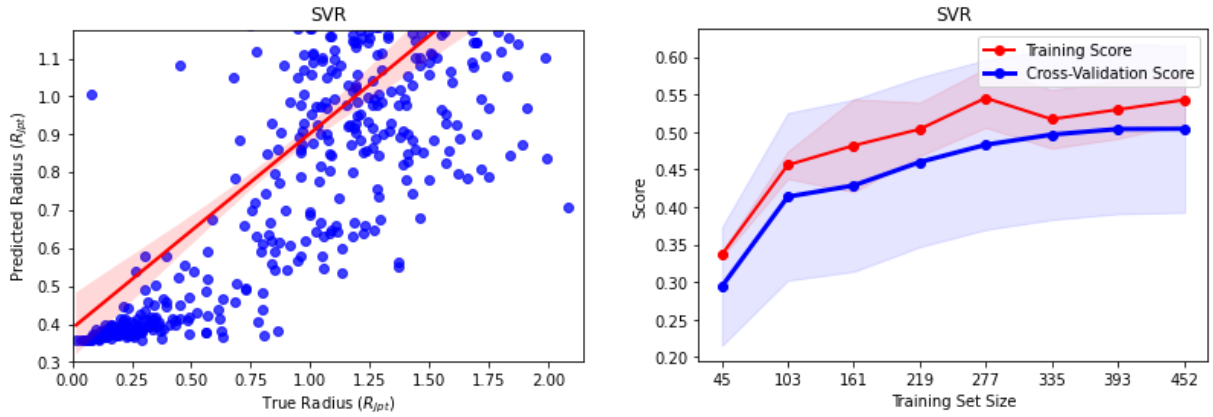


Figure 9:Plots of the predictive performance of the RBF kernel SVR model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.4.4 Sigmoid Kernel

The Sigmoid Kernel achieves an RMSE score of 0.33 and a cv score of 0.45. By using all the features, we acquired the following plots. We observed that the behaviour of the Training Score has no general agreement with the Cross-Validation Score, and we have a case of very bad

fit. We can also note that the standard deviation of the cross-validation is quite broad which only worsens the situation.

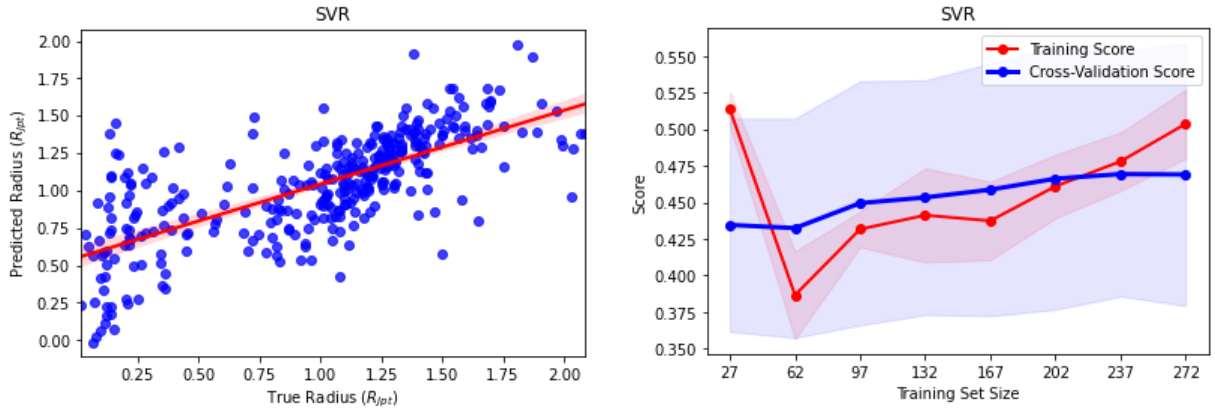


Figure 10: Plots of the predictive performance of the Sigmoid Kernel model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.4.5 Linear Kernel

Finally, the Linear Kernel ,using all the features, achieves a score of 0.32, close to the Sigmoid Kernel mentioned above and a cv score of 0.36. As far as the plots are concerned, again we observe very low scores overall. As expected we have a situation similar to the LinearSVR.

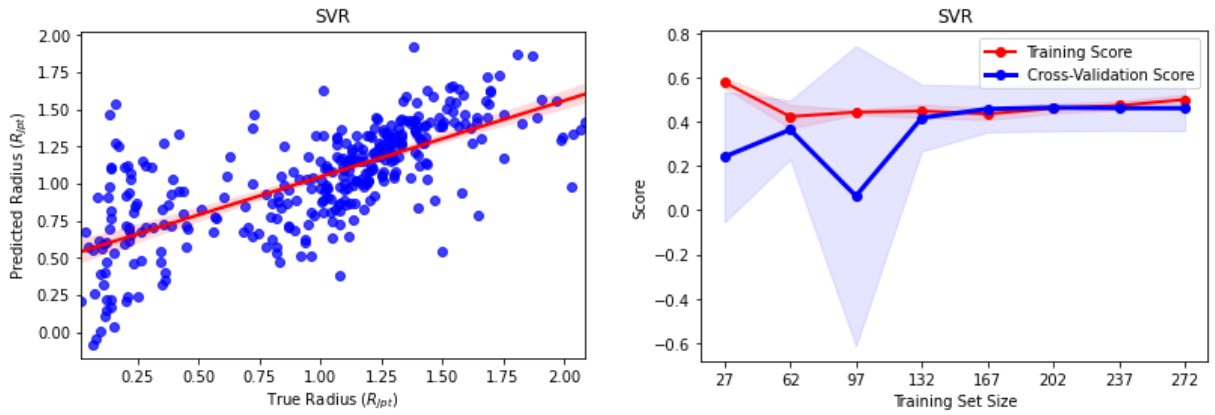


Figure 11:Plots of the predictive performance of the Linear Kernel model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.5 Ensemble Methods

Finally we turned to ensemble methods, with hopes of getting better predictions for the exoplanet's radius. An ensemble method relies on the joint effort of various weaker predictors in order to create a model with greater accuracy. We tried methods such as Random Forests, Extra Trees, AdaBoost and Gradient Boosting.

2.5.1 Decision Trees

All of the following ensemble methods rely on the decision tree algorithm which we implemented first. We used the CART (Classification and Regression Tree) algorithm as implemented in the Scikit-Learn library. The algorithm splits the training set in two subsets using a single feature k and a threshold t_k. Deciding on the purest split is done by minimizing the cost function that is reliant on the MSE of the predicted feature (Géron 2017, 305-306). It splits the dataset continually, according to the regularization we apply to the model, creating new nodes and leafs at each split, until it stops when it reaches a certain maximum depth. Our implementation, after optimization with `GridSearchCV`, was able to get an RMSE of 0.1 and a cv score of 0.769.

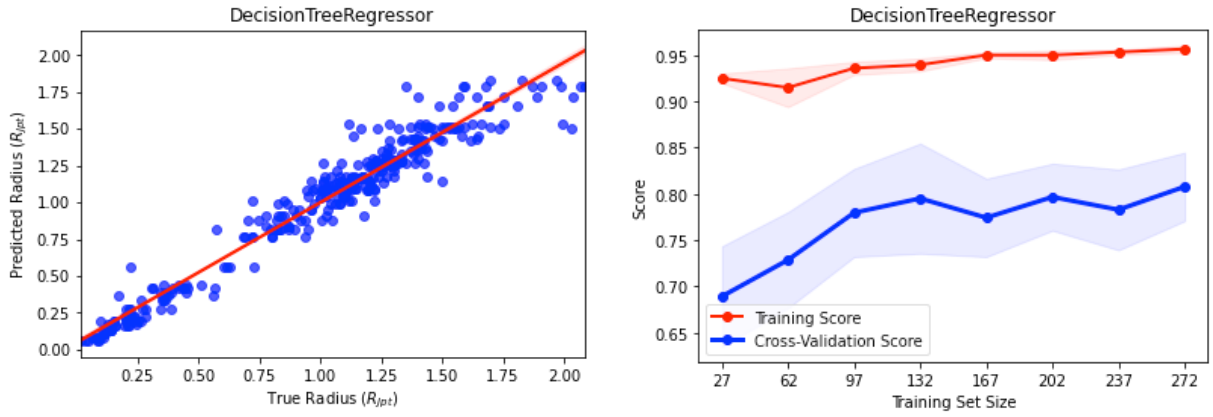


Figure 12: Plots of the predictive performance of the Decision Tree model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

Based on the learning curves the decision tree is overfitting on the training set, but has relatively low bias since it is able to extrapolate relations in our data and get a good cv score. The gap between the training score and the cv score is indicative of high variance, although the learning curves seem to begin to converge, meaning that we might need more data to be certain. We implemented and optimized DT's because the AdaBoost method, that we will discuss later, performs significantly better when we use already regularised, better performing models.

2.5.2 Random Forests

A random forest works by training multiple decision trees with a random subset of the training data and making a prediction by aggregating the prediction of all the predictors in the forest (Géron 2017, 317). Aggregating is done by taking the *average* of the predictions. The

random forest algorithm was fitted using both `RandomizedSearchCV` and `GridSearchCV` methods to cover as large of an area of the parameter space as possible without hindering training time too much (average training time, even with conservative parameter ranges on the optimization methods, was about 5-6 minutes). The resulting model was the best performing model so far with an RMSE of 0.08 and a cv score of 0.841.

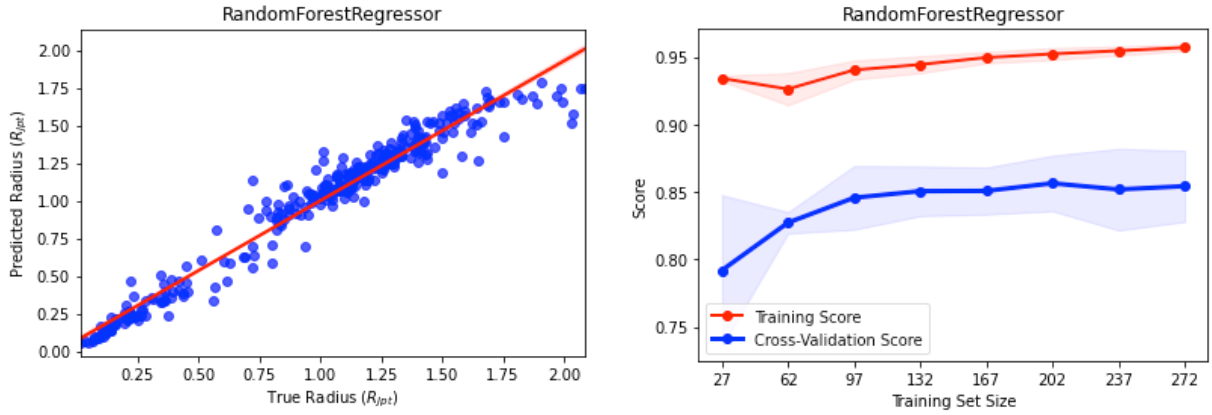


Figure 13: Plots of the predictive performance of the Decision Tree model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

As we can see from the training curves in Fig 13. the model performs better on the training set than the validation, insinuating that the model is overfitting. The gap between the two curves indicates that the model has high variance and it can be improved by either increasing the training set size, or constraining the hyperparameters, or both. Since it already has passed a rigorous optimization process the model needs to see new training data to improve its performance.

2.5.3 Extra Trees

The Extra Trees (Extremely Randomised Trees) algorithm works very similarly to the Random Forest algorithm but instead of choosing the best possible thresholds for each feature, it chooses them randomly, making them computationally faster (Géron 2017, 322-323). This technique theoretically trades more bias for lower variance (Geurts, Ernst, and Wehenkel 2006). After optimization the algorithm got an RMSE of 0.1 and a cv score of 0.77.

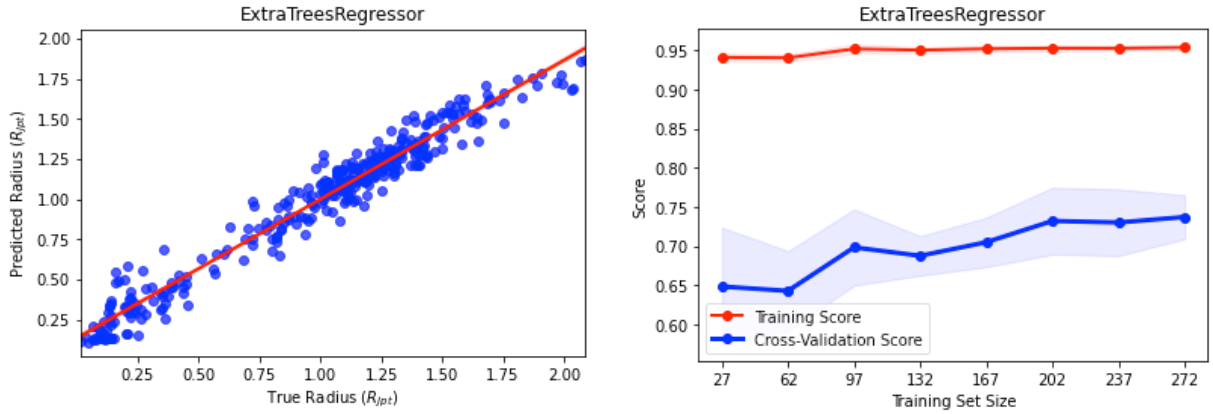


Figure 14: Plots of the predictive performance of the Decision Tree model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

As we can see the decrease in variance did not make a relevant difference in the overfitting behaviour of the algorithm, but the increase in bias worsened its performance on the validation set. It is important to note that by default the Extra Trees algorithm, as provided in the Sci-kit learn library, has the `bootstrap` parameter set to `False` making the algorithm sample elements of the dataset *without replacement*, which means that the training data each predictor in the ensemble sees is greatly decreased and thusly making the predictive performance of the model lacking. We eventually decided to set it to `True`.

2.5.4 AdaBoost

The AdaBoost algorithm is a boosting algorithm, meaning it combines multiple weak learners by training them sequentially and having each one trying to correct its predecessor (Géron 2017, 325). After each predictor is trained it is attributed a *predictor weight* based on its error rate, which is later used to adapt the instance weights of the next predictor. Our implementation uses the decision tree with the optimized hyperparameters we analysed in section 2.5.1 as a base model, so we can cut time from optimizing adaboost since it is considerably slower than other algorithms because of its non-parallelizable nature. We focused our optimization to the `n_estimators`, `learning_rate` and `loss` hyperparameters. The model underperformed when compared to the random forest, but did better than the extra trees model and the decision tree base model, with an RMSE of 0.06 and a cv score of 0.82. AdaBoost shows a performance akin to the forest - based regressors from the previous section, by overfitting on the training set and thus presenting a low bias but high variance.

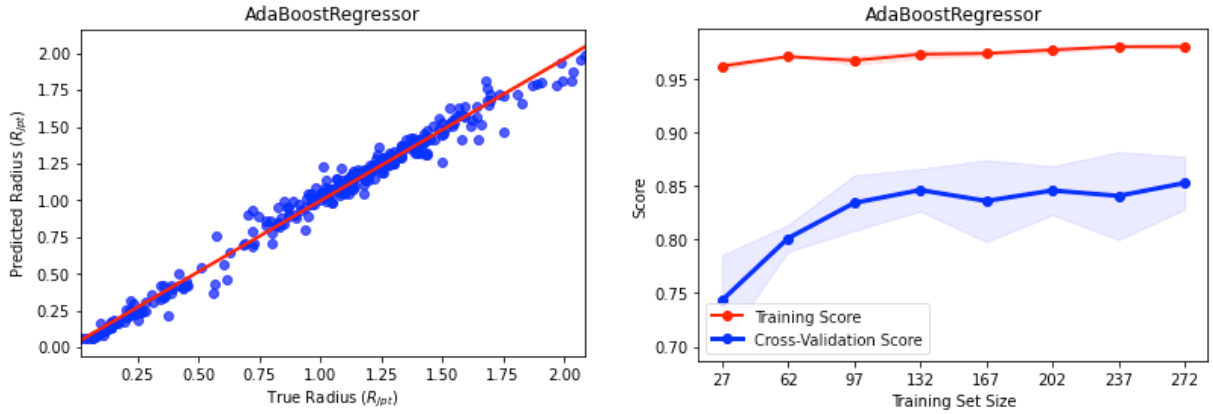


Figure 15: Plots of the predictive performance of the Decision Tree model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

2.5.5 Gradient Boosting

Finally we implemented a GBRT (gradient boosted regression tree) algorithm which, like adaboost, sequentially adds new predictors to the ensemble, but instead of tweaking the instance weights of each new predictor, it tries to fit the new predictor to the *residual errors* of the previous one. The implementation was relatively successful, with an RMSE of 0.13 and a cv score of 0.83.

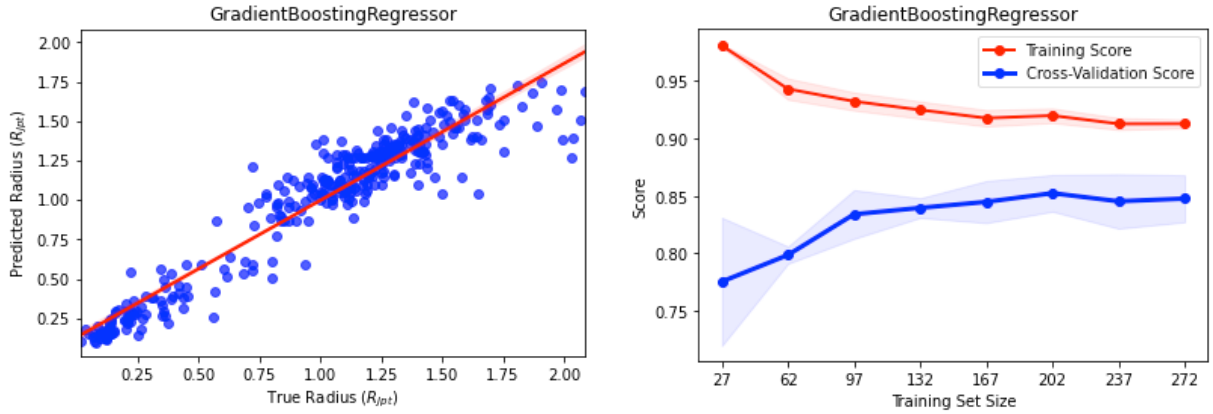


Figure 16: Plots of the predictive performance of the Decision Tree model. *Right:* predicted radius versus the "true radius" (the label as in the training data). The radii are compared with the 1:1 line in red. *Left:* mean cv-score of the model as a function of the training set size.

From the two boosting regressors we implemented, the GBRT gets a lower RMSE and as we can see from the learning curves it is able to get a low bias and the lowest variance of all of our regressors so far. We believe that if we had more training data the GBRT would be the best performer in terms of cross validation performance.

3. Results

3.1 Performance

Our test set was composed of 114 samples which as we mentioned were chosen by the `train_test_plit` function. We avoided exploring the test set after creating it to avoid introducing any bias to the models. After evaluation the best performing models so far have been the random forest and the GBRT we analysed in sections 2.5.2, 2.5.5. Their performance on the test set is similar with the RF getting a R^2 score of 0.86 and the GBRT getting a score of 0.85. They both got the same RMSE of 0.18. Comparatively the two models have similar performance, but we chose to use the RF for our predictions.

3.2 Predictions

As an example of how well the model generalizes we wanted to examine its predictions on the solar system planets . The solar system's planetary data is a good test set, since there has been extensive research done to get extremely accurate values for each feature[2]. The radii the model predicted can be seen in Table 3 in the Appendix.

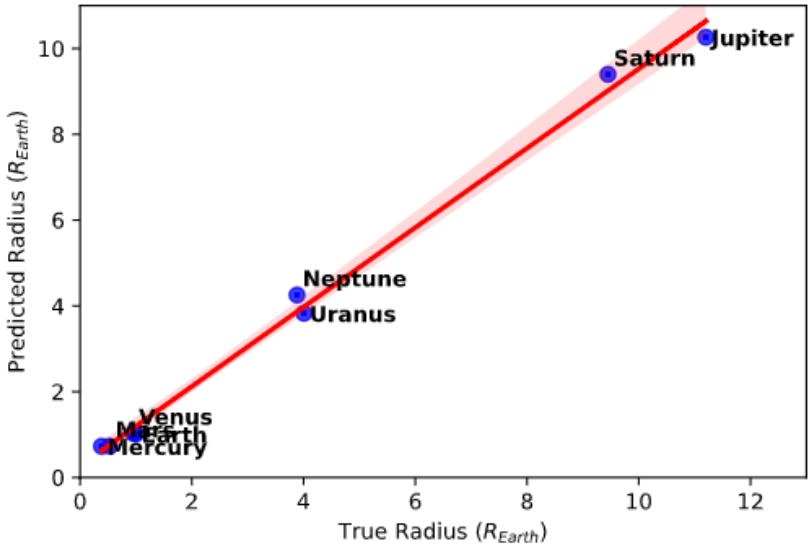


Figure 17: Plot of predicted radius of the solar system planets versus their real radius. The red line corresponds to a perfect fit and the shaded area represents the standard deviation of the data from the line.

Observing Fig 17. it seems that the RF model gets very accurate results. For Earth the model predicts a radius of 1.01 $R_\oplus$, which is almost a perfect fit. On the other hand we get a predicted radius of 10.3 $R_\oplus$ for Jupiter which is 0.9 $R_\oplus$ smaller than the real radius, which is larger than the RMSE of our model.

Another good visualization method we can use to gauge the performance of our model is a Mass-Radius-Equilibrium Temperature graph.

[2] "Planetary Fact Sheet - Ratio to Earth - the NSSDCA - NASA." 21 Oct. 2019, https://nssdc.gsfc.nasa.gov/planetary/factsheet/planet_table_ratio.html. Accessed 12 Dec. 2020.

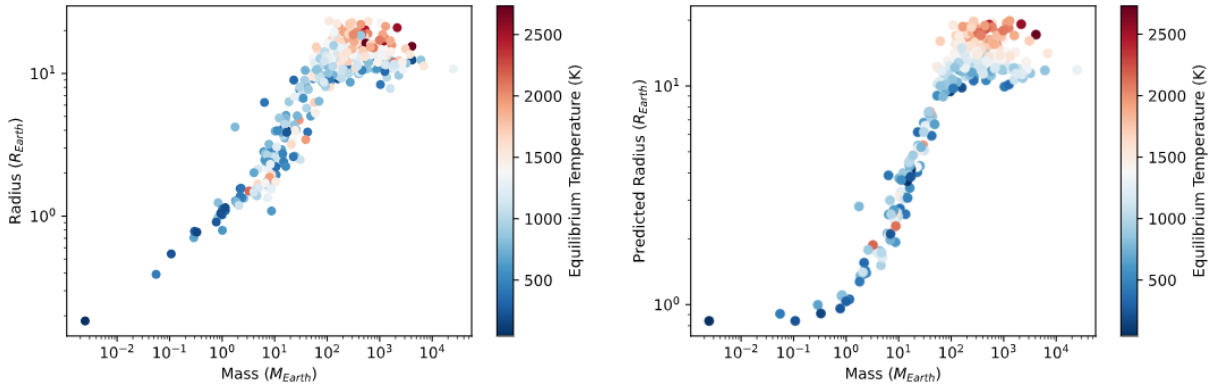


Figure 18 Mass - Real/Predicted Radius - Equilibrium Temperature graphs.

As we can see from the graphs the graphs RF is able to identify the behaviour of the relation between the relevant variables of the graph, but seems to underperform on predicting the radius of planets with small masses. Most planets in our dataset are massive between 10^2 and 10^4 $R_\oplus$ meaning the miscalculation of jupiter's mass might be an outlier.

4. Conclusions

We created and tested various supervised machine-learning models to try to estimate the radius across a wide range of masses, something that cannot be done by most physical models in the scientific literature. After a rigorous optimization and evaluation process we created a Random Forest model and a Gradient Boosted Regression Tree model that were able to get accurate predictions of radius. We compared the models predictions with data from both the test set and the Planetary Fact sheet for the solar system's planets and confirmed the accuracy of the Random Forest model.

5. References

[1] Ulmer-Moll, S., N. C. Santos, P. Figueira, J. Brinchmann, and J. P. Faria. "Beyond the exoplanet mass-radius relation." *Astronomy & Astrophysics* 630 (2019): A135.

[2] Aurélien Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems,O'Reilly Media ,2017

[3] "Introduction to Support Vector Machines (SVM)." n.d. geeksforgeeks. https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/?ref=rp.

[4] "Tutorial: Learning Curves for Machine Learning in Python." n.d. Dataquest. https://www.dataquest.io/blog/learning-curves-machine-learning/.

[5] "Open Exoplanet Catalog." n.d. Open Exoplanet Catalog. http://www.openexoplanetcatalogue.com/.

[6] Geurts, P., D. Ernst, and L. Wehenkel. 2006. "Extremely randomized trees." *Mach Learn* 63:3-42. https://doi.org/10.1007/s10994-006-6226-1.

[7] Drucker, Harris. "Improving regressors using boosting techniques." In *ICML*, vol. 97, pp. 107-115. 1997.

[8] Williams, David R. n.d. “Planetary Fact Sheet - Ratio to Earth Values.” Planetary Fact Sheet. https://nssdc.gsfc.nasa.gov/planetary/factsheet/planet_table_ratio.html.
[9] Laughlin, Gregory, and Jack J. Lissauer. "Exoplanetary Geophysics--An Emerging Discipline." *arXiv preprint arXiv:1501.05685* (2015).

Appendix: Tables

Table 3. Real and Predicted radii of the solar system planets from the Random Forest Model.

Planet	***Real Radius ($R_\oplus$)***	***Predicted Radius ($R_\oplus$)***
Mercury	*0.1915*	*0.727877*
Venus	*0.9499*	*1.030909*
Earth	*1*	*1.013470*
Mars	*0.533*	*0.729198*
Jupiter	*11.2*	*10.263932*
Saturn	*9.449*	*9.397807*
Uranus	*4.007*	*3.827436*
Neptune	*3.883*	*4.253674*