Machine Learning in Physics project: Success in Movies Phase 2: Traditional Models

Ali Setareh Kokab , Reyhane Ghanbari

Department of Physics, Sharif University of Technology

May 8, 2021

Contents

1 Summary

In phase two of our project, we tried five classic models for our regression problem, predicting the success of movies. The models which we tried are KNN[1], polynomial regression, and SVR[2] with two different kernels, namely RBF and polynomial, and the last model is random forest. For each model, we plot the learning curve and validation curve. In table 1 you can see the summary of these models' performances.

2 Preparing the data

Our goal is to predict the domestic gross of the movies **before** they release. As a result, we should only keep the available features before the date a movie goes on screen. These features are budget, opening theaters, running time (min), genre, age rating, day, month, language, country, and the rank of the movie's cast. The system of ranking is explained in the previous article. Our prediction variable (Y) is domestic gross. We take 2019 movies as our test data and take the rest as our training data set. We used PCA[3] to reduce the dimension of our data. As you can see in fig 1, 5 features are enough to explain our data. We worked with the 5 most important features in the entire of this article. We also use the logarithm of domestic gross and budget instead of their values due to the high skewness of these two columns.

[1]K-nearest neighbors
[2]support vector machine
[3]principal component analysis

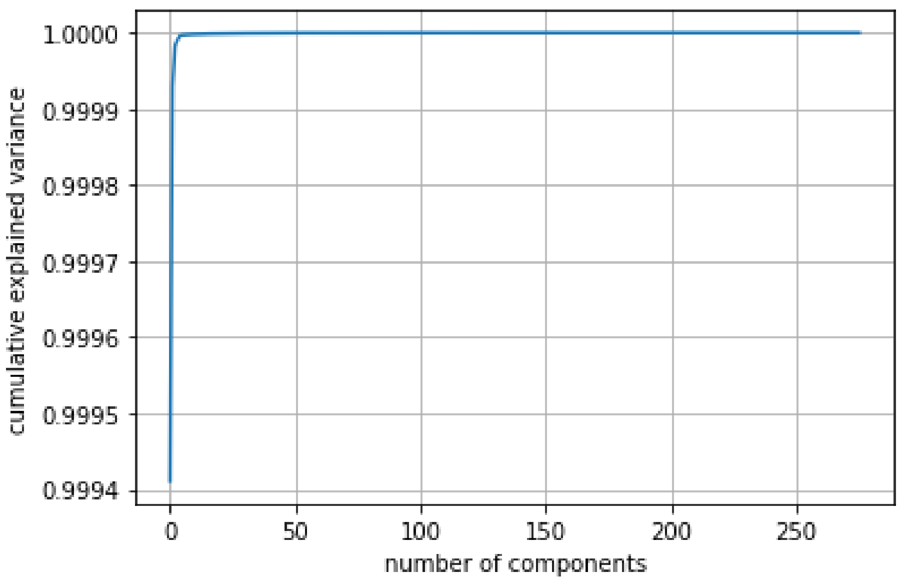

Figure 1: Cumulative variance versus number of components for PCA analysis. Evidently, more than 99 percent of the variance of the data can be explained by 5 features.

3 Metrics

we use mean squared error and R^2 (1) score to evaluate our models. These are widely used metrics for regression problems. R^2 is a **relative** measure of how well the model fits dependent variables and measures how the model can explain much variability in the dependent variable. Its value is between 0 and 1, and a bigger value indicates better performance.In eq 1, $\hat{y}$ and $\overline{y}$ represents the predicted and real values, respectively.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y_i})^2}{\sum(y_i - \overline{y})} \tag{1}$$

We also used mean square error (2) to evaluate the performance of our models. Despite the R^2 score, mean squared error is an absolute measure of the goodness of our fit. So it is a useful metric to compare different regression with each other.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2 \tag{2}$$

4 models

This part describes the regression models that we used on our data.

4.1 k-nearest neighbors

We used KNeighborsRegressor from sklearn for this part. The complexity of KNN models is determined by the number of neighbors that the regressor uses to estimate the best fit. To find the best number of neighbors, we plot the validation curve for this model fig 2. To find out the number of neighbors in which the distance between the validation curve and the training curve are lowest, we plot the difference between these two (fig 3) and find the minimum. As it is seen, the best result obtains with k = 600.

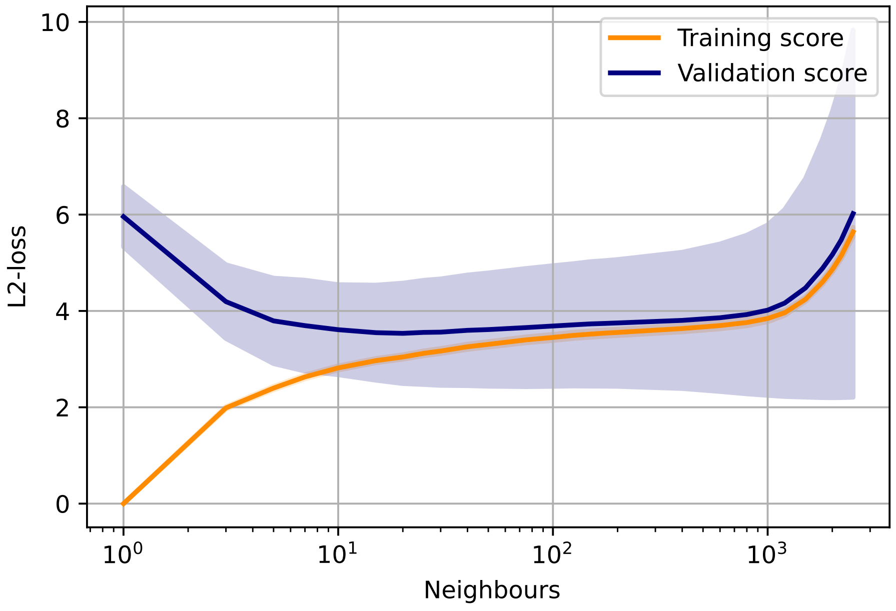

Figure 2: Validation curve for KNN model.

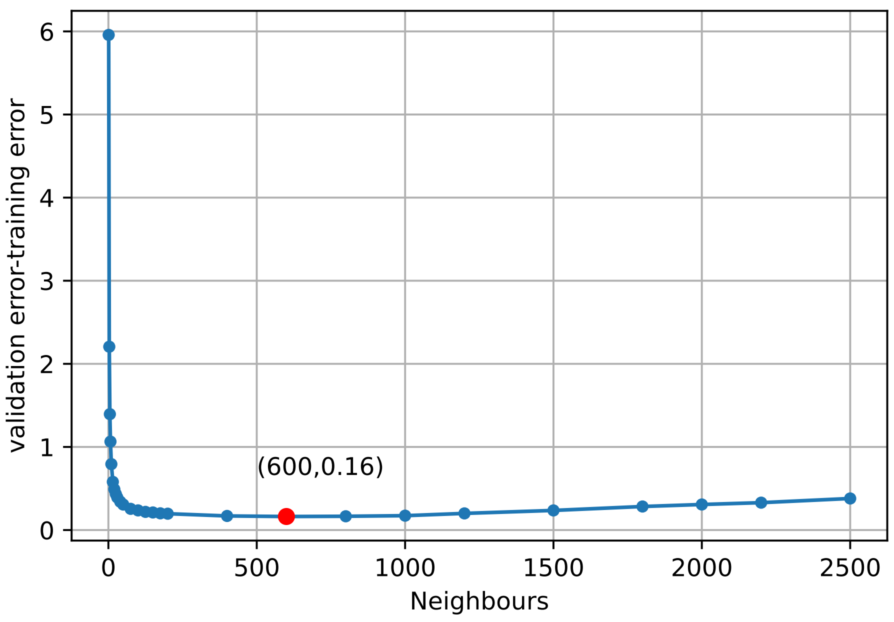

Figure 3: Difference between validation error and training error. The red point represents the optimum value.

After finding the best number of neighbors, we plot the learning curve fig 4. As is can be seen, about 2500 samples are enough to fit the model. The bias and variance of this model are equal to 3.92 and 0.1, respectively.

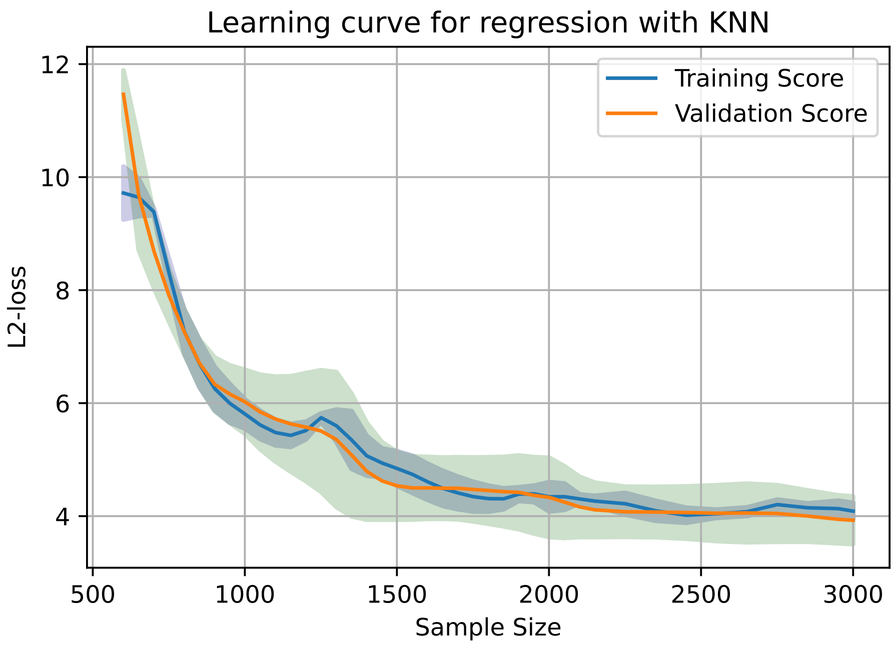

Figure 4: Learning curve for KNN with k=600

4.2 Polynomial regression

We used LinearRegression from sklearn for this part. The complexity of polynomial models is determined by the polynomial's degree to fit the data. To find the best degree of the polynomial, we plot the validation curve for this model fig 5. As it is seen, the best result obtains with a second-degree polynomial.

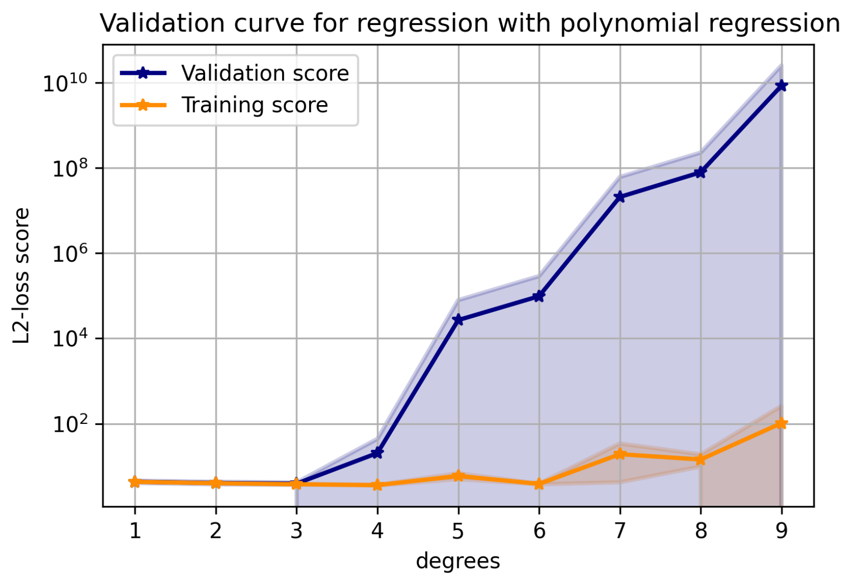

Figure 5: Validation curve for polynomial model.

After finding the best degree, we plot the learning curve fig 6. As it can be seen, about 2000 samples are enough to fit the model. The bias and variance of this model are equal to 3.39 and 0.1, respectively.

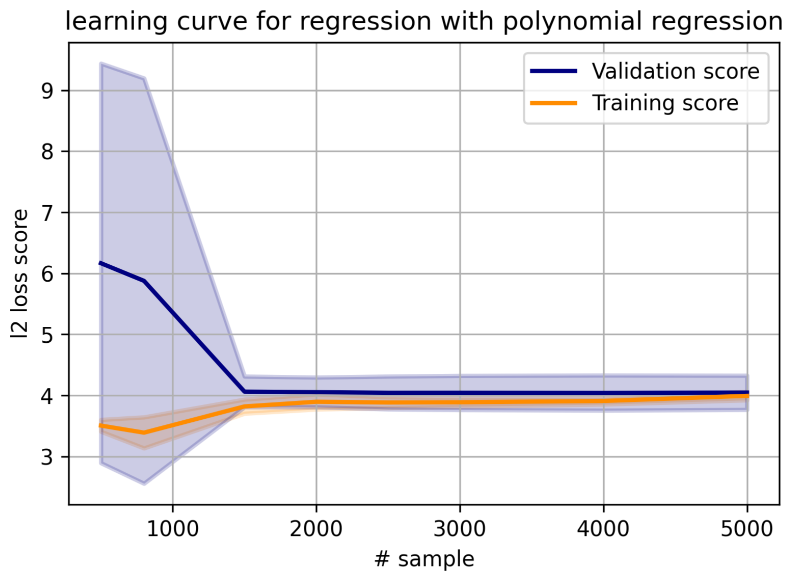

Figure 6: Learning curve for polynomial model with second degree.

4.3 SVR with RBF kernel

we used SVR regressor with RBF kernel from sklearn for this part. SVR with RBF complexity is determined by gamma parameter that indicates how far the influence of a single training example reaches. The gamma parameter can be seen as the inverse of the radius of influence of samples selected by the model. Another parameter in SVR is the C parameter that is the regularization parameter. We first explore these parameters separately by fixing one of them and changing the other fig 7 and 8. After that, we used grid search to find the optimum values for these parameters simultaneously. The optimum values are gamma = 0.00001 and C =10. You can see the learning curve in fig 9. AS you can see, about 4500 samples are enough to fit the model. The bias and variance are 3.7 and 0.3 , respectively.

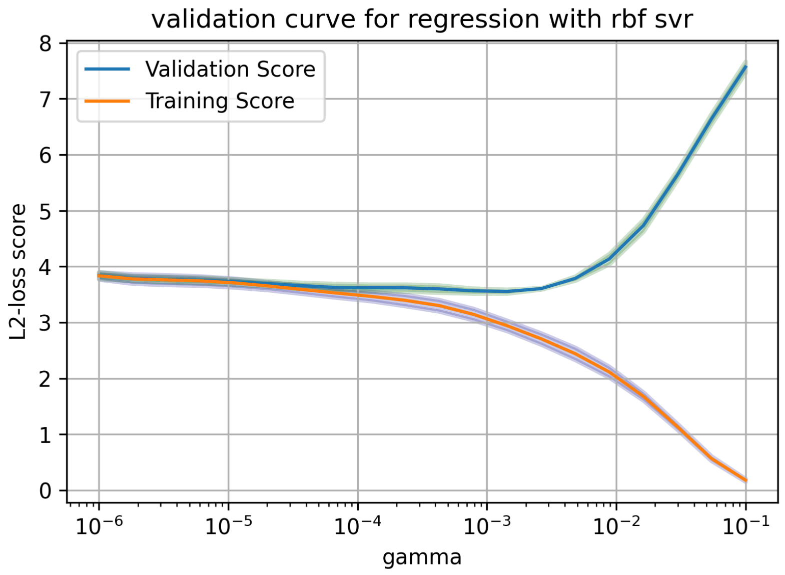

Figure 7: Validation curve for SVR with RBF kernel with C=10

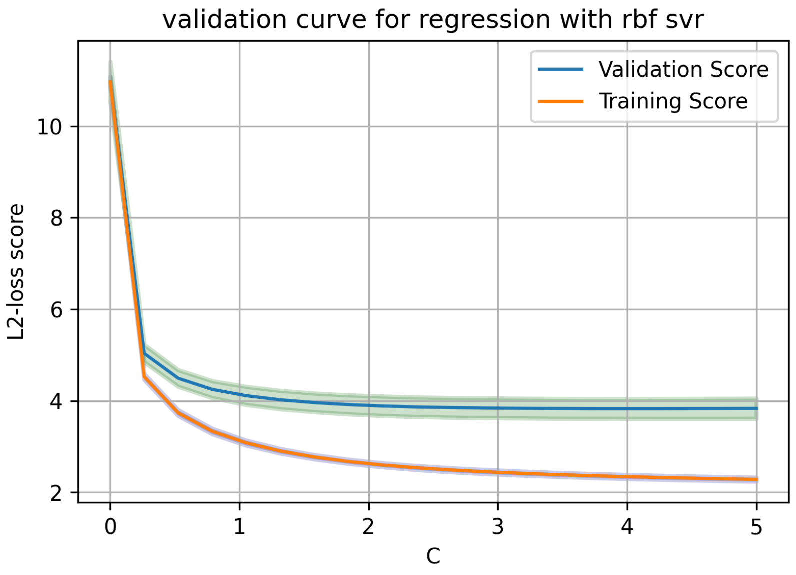

Figure 8: Validation curve for SVR with RBF kernel with gamma=0.00001

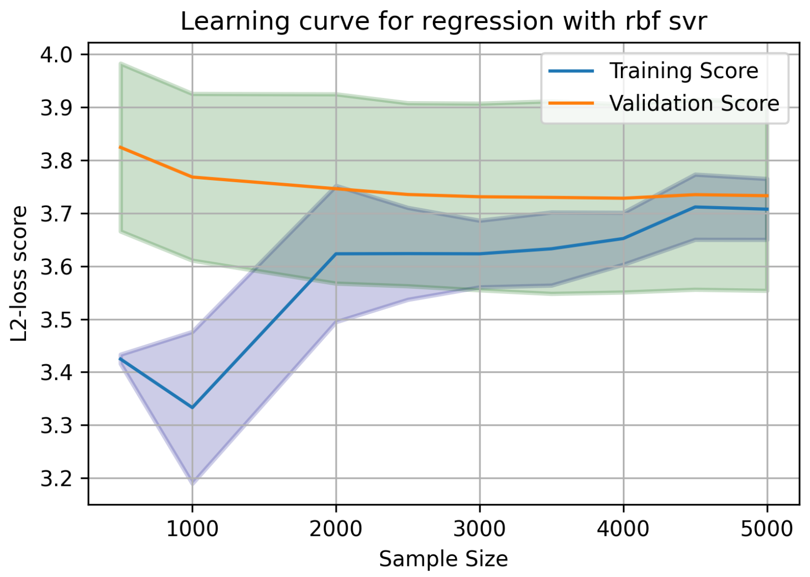

Figure 9: Learning curves for SVR with RBF kernel with gamma = 0.00001 and C = 10.

4.4 SVR with polynomial kernel

we used an SVR regressor with a polynomial kernel from sklearn for this part. SVR with polynomial kernel complexity is determined by the degree of the polynomial and regularisation parameter C. Another parameter in SVR is the C parameter that is the regularization parameter. We used grid search to find the optimum values of these parameters simultaneously. The optimum values are degree =1 and C = 2. You can see the learning curve in fig 10. AS you can see, about 3000 samples are enough to fit the model. The bias and variance are 4.35 and 0.2 , respectively.

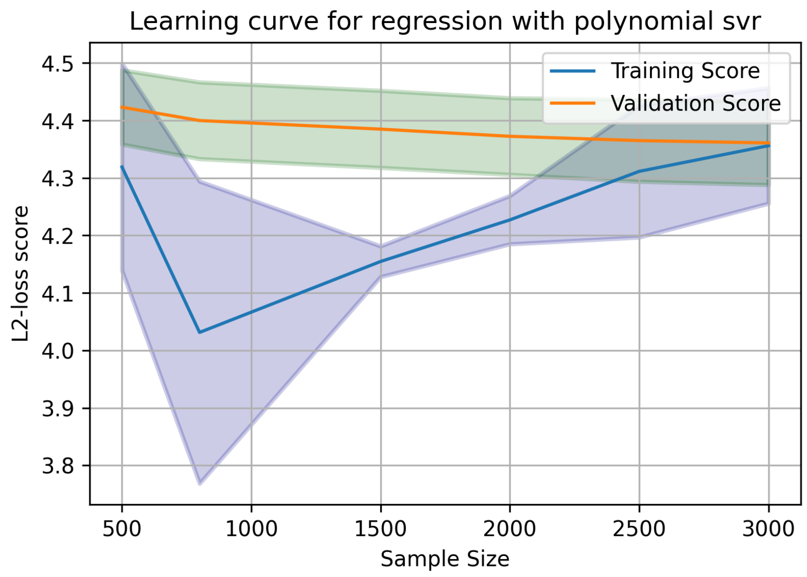

Figure 10: Learning curves for SVR with polynomial kernel with degree=1 and C=2.

4.5 Random forest

We used RandomForestRegressor from sklearn for this part. Random forest complexity is determined by the number of trees and the maximum depth of the trees. First, we plot the validation curve for a constant number of trees equal to 10 fig 11. Note that in this plot, both validation and train loss decrease by increasing the depth of the tree. Because of this, to pick the optimum value for the depth of the trees, we look at the difference between these two curves and choose the least depth that this becomes constant, which is the flat part of the curve in fig 12. As it is seen, the optimum depth for this number of trees is about 30. At this point, we get R^2 score of 0.66 and a mean squared error of 2.62 on the test data. In fig 13 you see the learning curve for this model. The bias and variance of this model are equal to 3.92.
As it was mentioned, random forests have two hyperparameters. In the previous part, we fixed the number of trees and played with depth. Now we use the sklearn grids search module to find the optimum depth and number of trees simultaneously. The optimum values for depth and number of trees are 10 and 190, respectively. For these values, the mean squared error is 2.42, and R^2 score is 0.67. As you can see, our performance is slightly better than the previous part. You can see the learning curve for these values in fig 14.

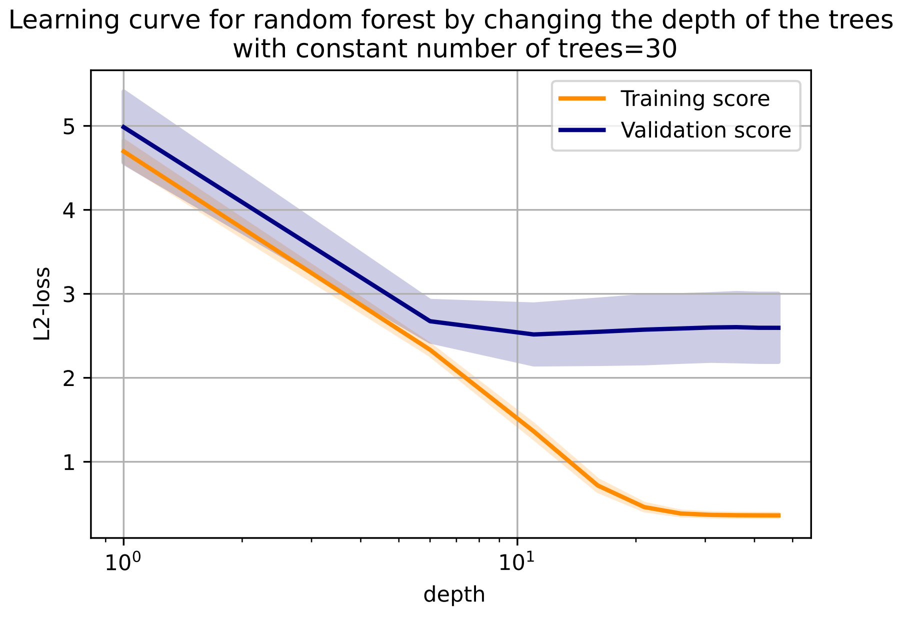

Figure 11: Validation curve for random forest model.

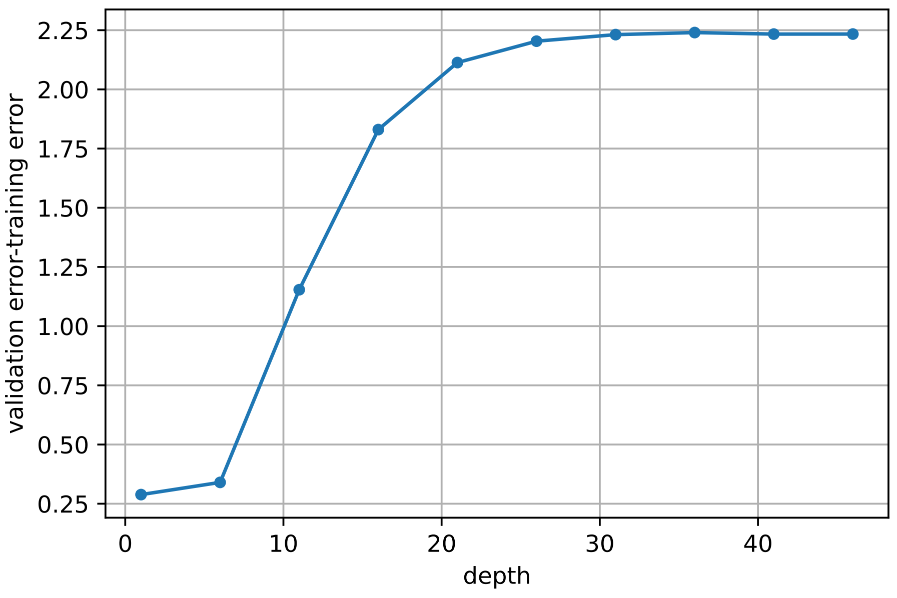

Figure 12: Difference between validation error and training error.

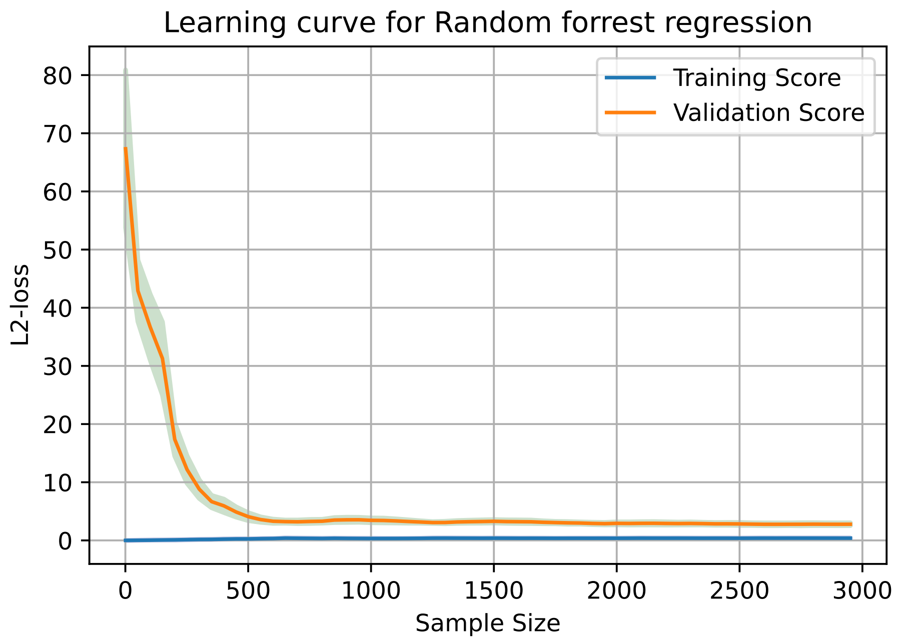

Figure 13: Learning curve for random forest 30 trees and maximum depth of 30

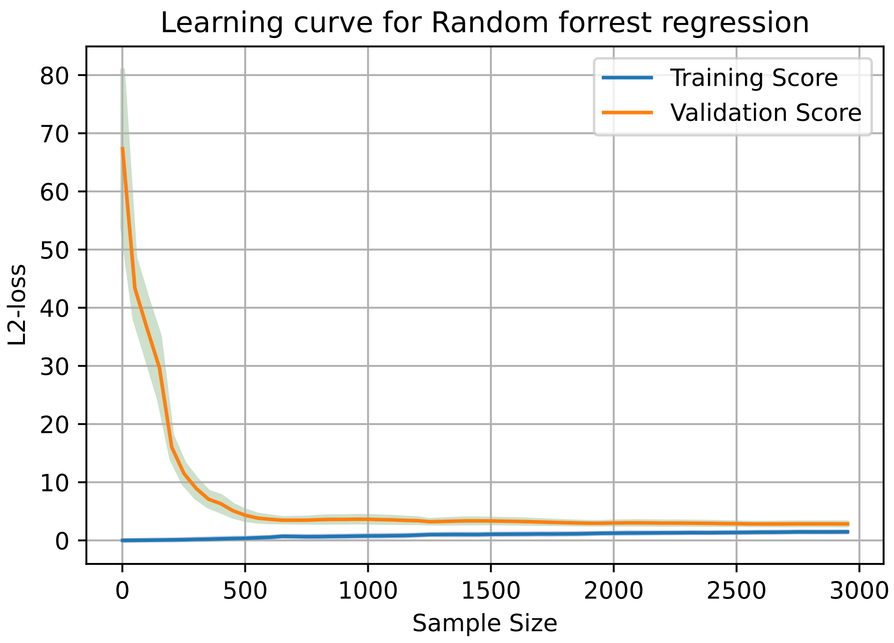

Figure 14: Learning curve for random forest with optimum values of 190 trees and maximum depth of 10. Note that in this figure the gap between validation and train is smaller in comparison with fig 13.

Model	MSE (train)	R^2 (train)	MSE (test)	R^2 (test)	training time	prediction time
KNN	3.69	0.42	3.12	0.46	699 ms	123 ms
Polynomial regression	4.00	0.58	3.39	0.53	11.6 ms	1.97 ms
SVR with polynomial kernel	4.34	0.58	3.72	0.48	104 ms	1.91 s
SVR with RBF kernel	3.70	0.65	3.4	0.53	3.13 s	271 ms
random forest	1.73	0.78	2.42	0.67	2.95 s	41 ms

Table 1: table of traditional models performance