09/12/2022

MACHINE LEARNING

1st Assignment

Leisure:
Applied Informatics

Department:
Introduction to Computer Science and Technology

Theme:"Regression problems»

Club:
1. Michael Zervas, ics20015
2. Nestoras Sotiriou, ics20012
3. Christos Drikos, ics20043

Academic year 2022-2023

Table of Contents

Charts Table of Contents

Introduction

The exercise is divided into 2 individual sub-problems which, however, have as a common purpose the identification of the best prediction function of the results. The differences between these two concern the knowledge or not of the model that generates the data. More specifically:

The first subproblem (1The part) is about finding the best approximation between 2 functions (one polynomial and one based on the sine and exponential functions) based on the noisy input data, knowing the parametric model. It is noted here that the model generating the data is the same sinusoidal function as before, with noise added.

The second subproblem (2The part) is to find the best possible approximation between 3 different linear regression functions (kNN, SVR, decision trees), based on the noisy input data generated before, this time without knowing the parametric model.

Methods applied

Language used: Python

1The part:

1The question)

To generate the 150 random numbers in [-4,4], which follow the normal distribution, the random.uniform() function from the numpy library was used, while the sort function was used to sort the results. These numbers will hereafter be referred to as inputValues.

2The question)

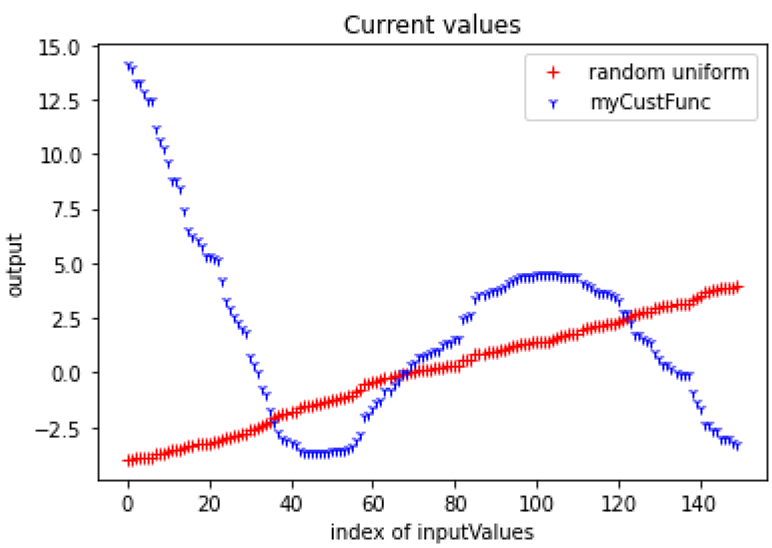
Similarly, for the implementation of the myCustFunc function, the ready-made functions sin() and exp() were used from the numpy library.

3The question)

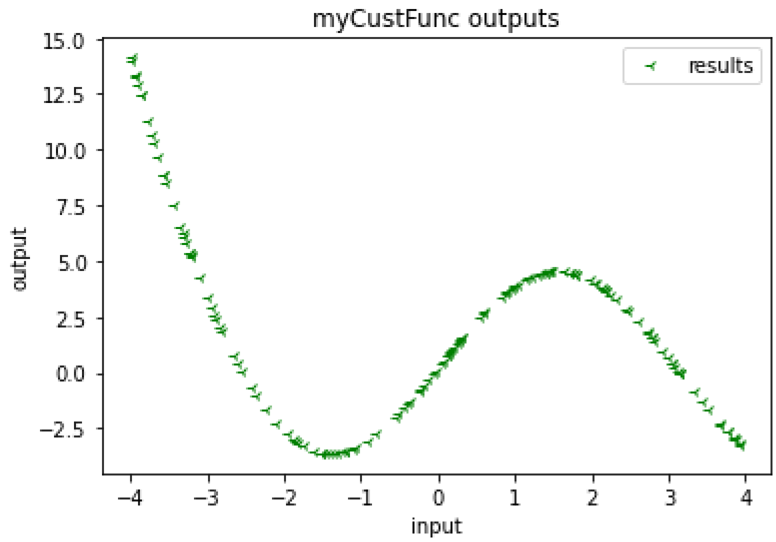
The values of λ are given randomly $_1$= 0.2, λ_2= 4.5 for the parameters of myCustFunc(), and inputValues is given as x. A new ndarray of 150 positions is produced.

4The question)

In this question, although it asked for one graph, we provide two. The reason is that we felt that it was not clearly specified whether the data should be displayed as a function of each other or not. This is how we ended up with the following two diagrams:



In the first diagram, the x-axis refers to the position of each element in the sorted array inputValues, while the y-axis shows the values of the corresponding positions of inputValues in red (which were created in question 1) and in blue the values that gives myCustFunc with input the value of the corresponding position each time of inputValues.



In the second diagram the x-axis shows the value given as input and the y-axis shows the results produced by myCustFunc for the corresponding data. The input data is the same as before (inputValues) but this diagram shows the input (x) and output (y) values better, as well as the relationship between them obtained through the myCustFunc function.

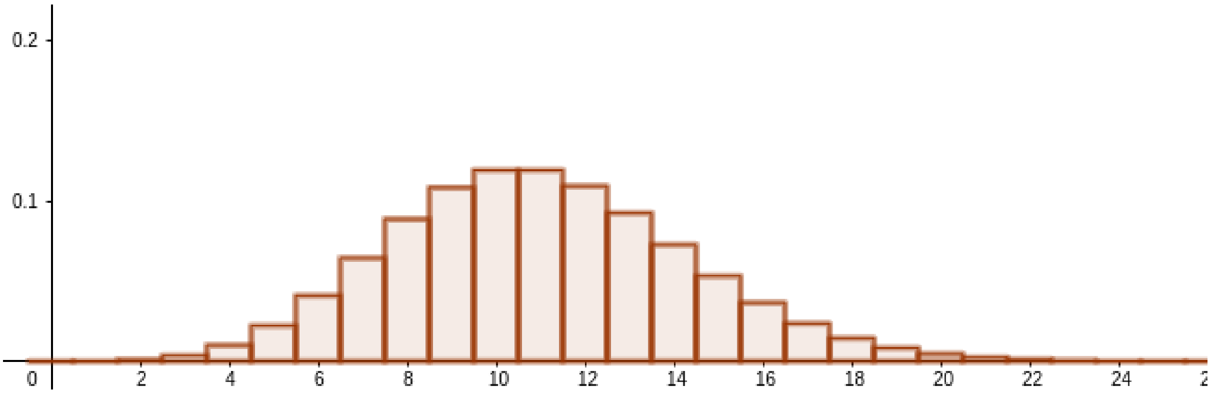
5Thequestion)

Noise was introduced based on the Poisson distribution. In each record of inputValues, a random value from the Poisson distribution with λ=11 was added to it. To achieve this, the corresponding function from numpy (random.poisson()) was used.

The Poisson distribution calculates how many times an event can occur in a certain time interval. It is a discrete function. More specifically for events with an expected distance λ the Poisson distribution f(k;λ) describes the probability of k events occurring within the observed interval λ. It is defined by the formula:

$$f(k;\lambda)=\frac{\lambda^k e^{-\lambda}}{k!}$$ and graphically has the general form:

We used λ=11 and its form is as follows:



6Thequestion)

Since we know the parametric model, here we are allowed to use scipy.optimize.curve_fit.

7Thequestion)

The polyval function is used to create the polynomial

numpy, which takes as arguments the unknown x and the parameters of x^nin ascending order for n=0 to n=4 (in the exercise). It is then converted to an ndarray via the numpy.array() command so that it can be used by curve_fit().

8Thequestion)

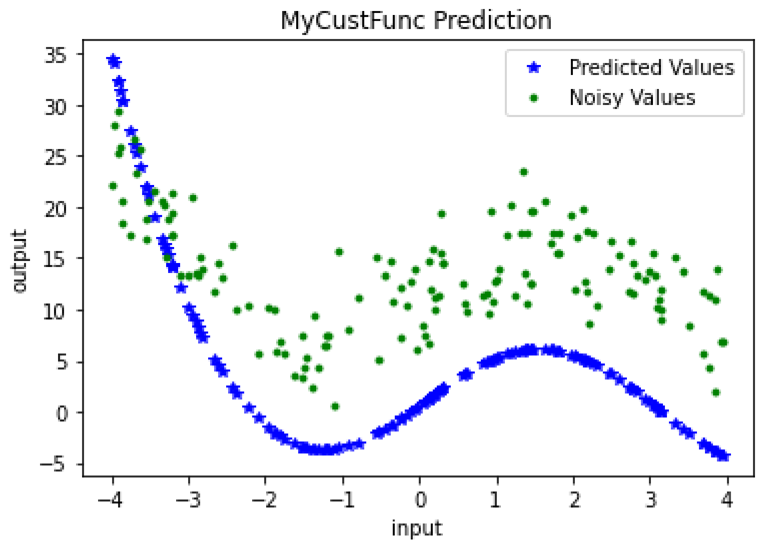
Using scipy.optimize.curve_fit, we approximate the optimal parameters of the polynomial.

9Thequestion)

Before making the graphs we have to calculate the outputs using the parameters we approached before and with the inputValues.
In the charts we have also included the Noisy Values to show how well each function predicts. Thus we have:

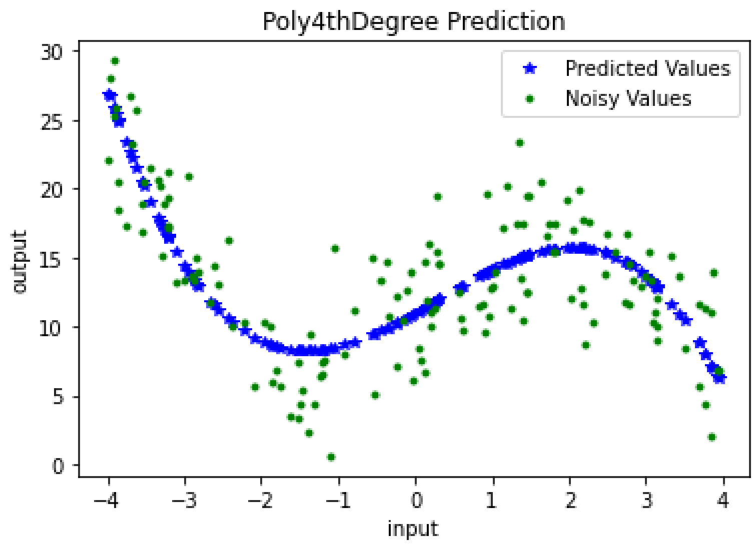
9a)

The diagram shows the values predicted by the myCustFunc function (blue color) and the Noisy Values which are essentially the values we would like our model to approximate. Here, however, we see that although at the beginning the Predicted Values of MyCustFunc are relatively good, when the input passes the value close to -3, the results start to deviate quite a bit from the Noisy Values. So we can conclude that myCustFunc is hardly at all accurate in this particular problem. As the noise increases, so will its accuracy and vice versa. In the case of minimal noise, myCustFunc could make better predictions because the function that generated the data is myCustFunc itself, so the noisy data would be almost the same as the noiseless one.



9b)

Here we see the corresponding predicted values of the polynomial function in relation again to the Noisy Values. We notice that this prediction seems more accurate than the previous one as the Predicted Values are better distributed among the Noisy Values. The polynomial function 4ᵤdegree seems to fit the noise data better.



10The question)

The ready-made functions of sklearn were used to calculate the required metrics.

We have:

	Mean Absolute Error (MAE)	Root Mean Square Error (RMSE)
original_values - noisy_values	11.38	11.83
noisy_values - myCustFunc	9.43	10.24
noisy_values - poly4thDegree	2.72	3.31

- The polynomial has approx67% lower MAE than myCustFunc
- The polynomial has approx71% smaller RMSE than myCustFunc

The goal is to achieve the lowest possible values for the metrics. Thus comparing the same metrics each time, we notice that the polynomial function finds much better results. The conclusions we drew from the charts are also verified by the metrics.

2Thepart:

2Thequestion)

The separation of the data into train, validation and test was done as follows:

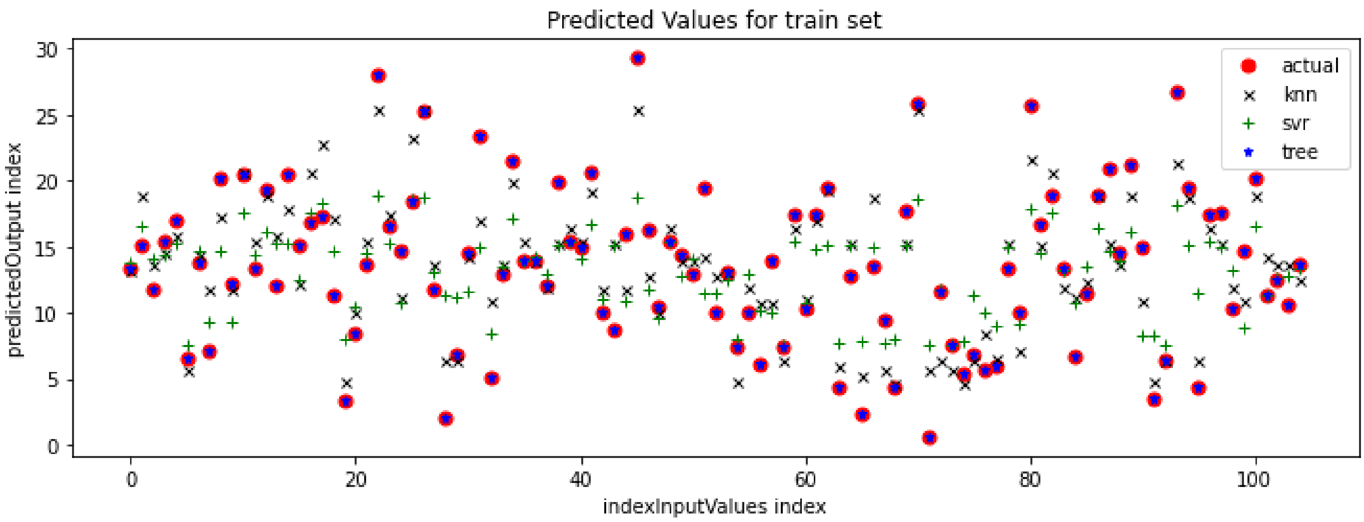
- 70% was used for train
- 7% was used for validation and
- 23% was used for testing

3Thequestion)

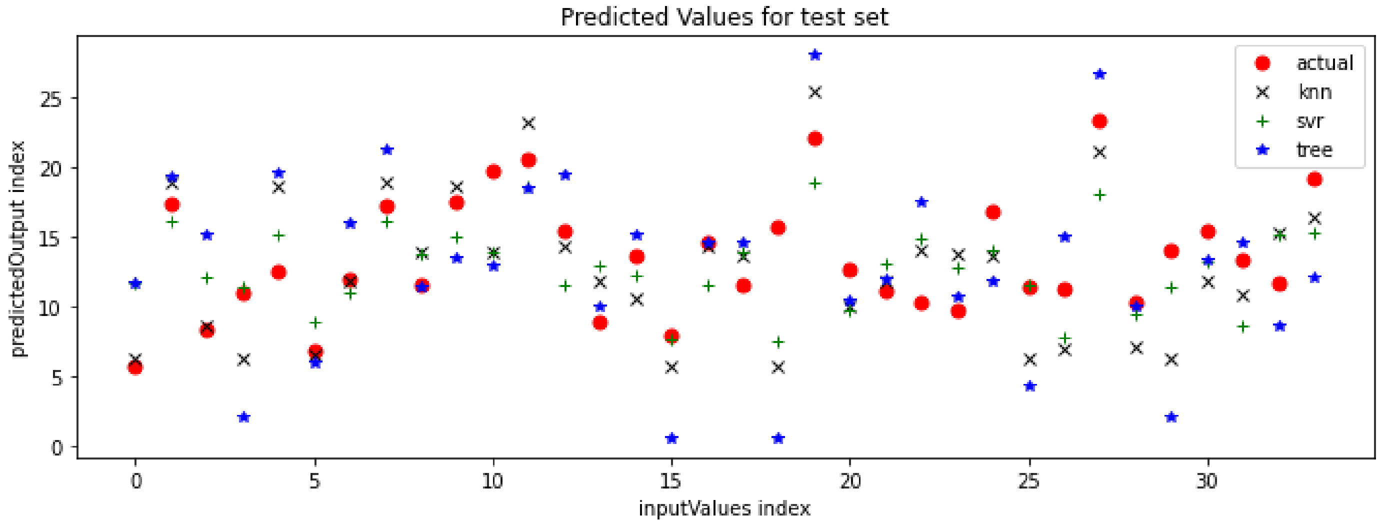
kNN, SVR and Decision Tree Regressors were used.
For the training of all regressors, the respective fit() functions of sklearn were used on the inputValues data with noisyInput, in the corresponding train sets that were created.
Furthermore, kNN is initialized with n=5 neighbors and SVR with a radial basis function (RBF) kernel.

4Thequestion)

To evaluate the performances, sklearn's predict was used, with the parameter inputValues corresponding to the train set.
In the first diagram we observe the accuracy of the Predicted Values of each regressor separately in relation to the actual values. We can see that the Decision Tree has 100% accuracy in the train set, i.e. it overfits the data. This is quite negative and will significantly reduce its performance on the test set. On the contrary, the other two regressors seem to perform equally well in the train set.



The second diagram shows how well each regressor performs on "unknown" data compared to the training data, i.e. the test set. As mentioned previously, we see that there are some results where the prediction of the decision tree is quite far from the normal value, something that does not happen to the same extent with the other 2 regressors.



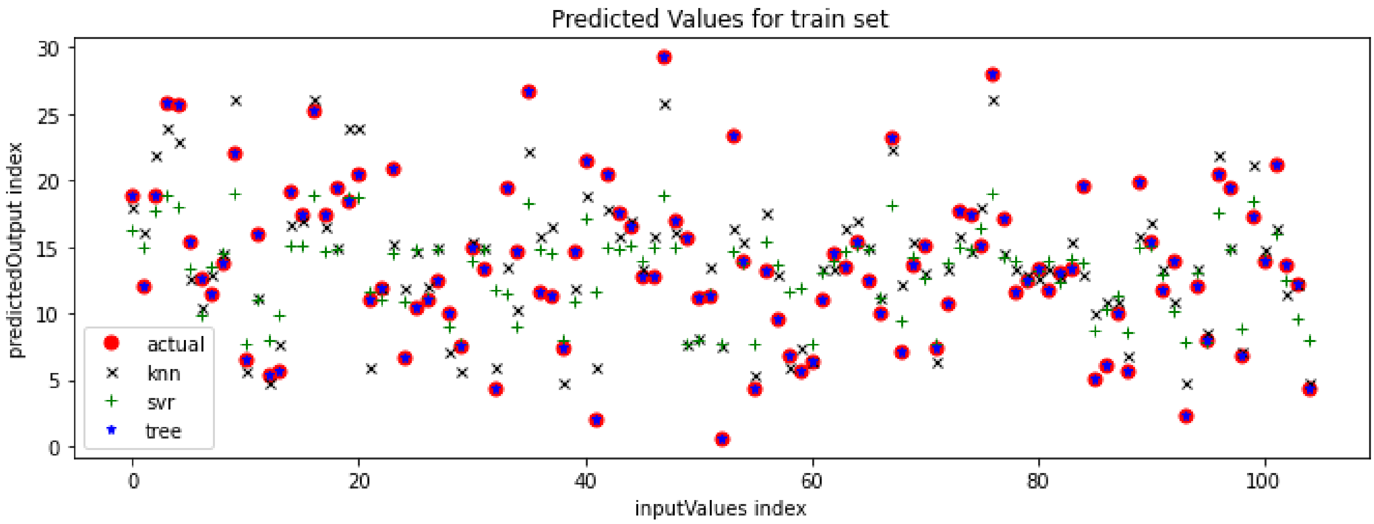
The metrics were calculated based on actual values and predicted values.

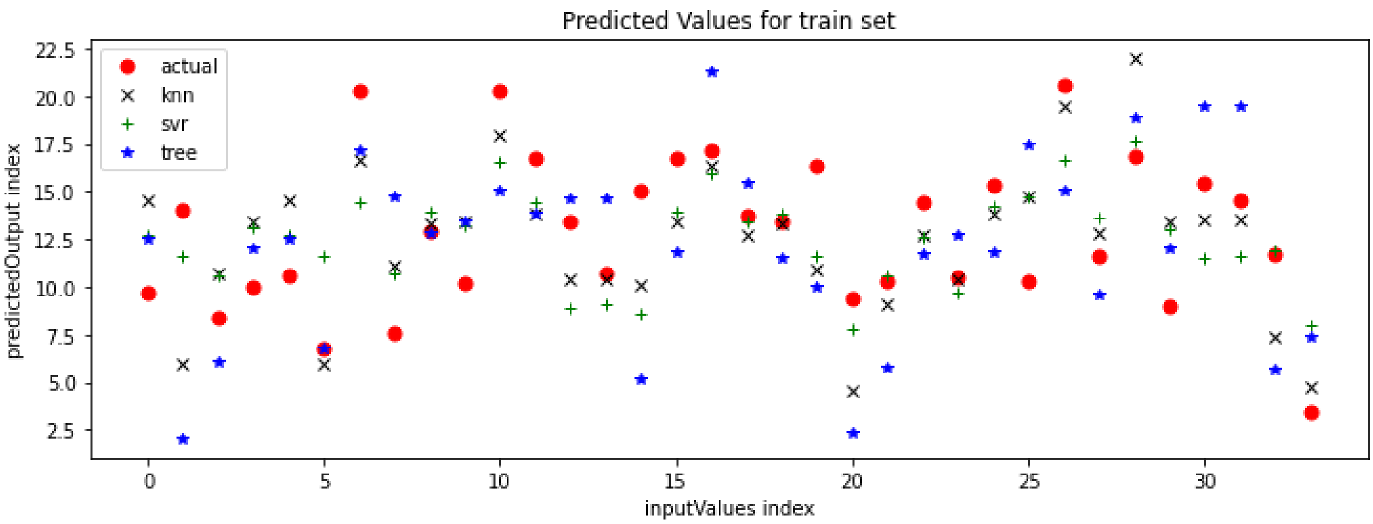
(test set)	MAE	RMSE	MAX ERROR
kNN (k=5)	3.00	3.67	9.97
SVR	2.90	3.38	8.09
Decision Tree	4.31	5.94	15.07

The MAX ERROR of decision tree is quite larger than kNN and SVR and this is due to overfitting. Consequently, the other 2 metrics are also affected, with RMSE being more affected because it "punishes" the outliers more, taking the squares of the distances each time.
We conclude that with the given data, SVR is more efficient than kNN and Decision Trees, with kNN being more efficient than Decision Trees only. But since kNN does not deviate much from SVR we could conclude that we can use both in this particular problem or we could also re-run it many times and collectively see which one is better.

5The question)

Normalization was done on inputValues in the interval [0,1] via MinMaxScaler.





(test set - normalized)	MAE	RMSE	MAX ERROR
kNN (k=5)	2.71	3.29	8.02
SVR	2.68	3.14	6.45
Decision Tree	3.97	4.72	11.95

After normalization we see how the 3 metrics and the 3 regressors have improved. This is because the normalization smoothed out the distances between the points. Of course, it does not mean that the values will decrease every time because the results depend on the random separation of the dataset into train, test and validation. In general, however, we can conclude that with normalization there is a greater chance of achieving a better approximation.

Conclusions

Regarding the first part, where the parametric model is known, the polynomial function 4$_{u}$degree performs much better than myCustFunc. The only case where myCustFunc would perform better would be if we minimized the noise.
In the second part, where the parametric model is unknown, both SVR and kNN perform equally well. Improvements could probably be seen if we found a better k in kNN and used a different kernel in SVR. With normalization they perform even better. Decision Tree on the other hand overfits the data and is therefore not efficient. The reason this happens is because the depth of the tree is likely to be large. In order to avoid this we can resort to 2 techniques. The first is pre-pruning where we create a tree with fewer branches than it should (by defining for example a max depth). The second is post-pruning where the complete tree is first created and then parts of it are removed.

Sources

https://www.w3schools.com/python/numpy/numpy_random_poisson.asp https://
www.geeksforgeeks.org/numpy-random-poisson-in-python/ https://numpy.org/doc/
stable/reference /random/generated/numpy.random.poisson.html https://
en.wikipedia.org/wiki/%CE%9A%CE%B1%CF%84%CE%B1%CE%BD%CE %BF%CE
%BC%CE%AE_%CE%A0%CE%BF%CF%85%CE%B1%CF%83%CF%8 3%CF%8C%CE%BD

https://mathworld.wolfram.com/PoissonDistribution.html
https://www.geogebra.org/m/PUS7ZYW8
https://link.springer.com/chapter/10.1007/978-1-4471-4884-5_9 https://
towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991