Albert-Ludwigs-University Freiburg | 79085 Freiburg im Breisgau | Germany

Department of Computer Sc.
Bioinformatics Group

Tool Resource Prediction for Genomic Datasets

Masterproject Report

Submitted by:
Öner Aydogan
Computer Science (Master)
Matriculation number: 4947195
aydogan@tf.uni-freiburg.de

Examiner:
Prof. Dr. Rolf Backofen

Supervisors:
Simon Bray
Anup Kumar

2022

“Tool Resource Prediction for Genomic Datasets”
Version from October 3, 2022

PDF-LaTeX 2_{ε}

Contents

1 Introduction **1**

2 Related works **3**

3 Background **5**

3.1 Random Forest 5

3.2 Extreme Gradient Boosting 7

4 Methods **9**

4.1 Galaxy job run dataset 9

4.1.1 Description of the dataset 9

4.1.2 Preprocessing 10

Reducing dataset size 10

Filtering faulty data 11

Removing outliers 11

4.2 Training and evaluation pipeline 12

4.2.1 Base estimators 12

4.2.2 Training and evaluating the estimators 12

4.3 Hyperparameter Optimization 13

4.4 Prediction with uncertainty 14

4.4.1 Prediction intervals 15

5 Results **18**

5.1 Removing faulty data 18

5.2 Removing outliers 19

5.3 Correlation analysis 19

5.3.1 Moderate-high correlation 20

5.3.2 Low-moderate correlation 24

5.4 Performance on other datasets 27
5.5 Hyperparameter Optimization & comparison to Galaxy baseline 30
5.5.1 Analysis on resource consumption 33
5.6 Accuracy-Failure trade-off 33

6 Discussion and Conclusion 35

A List of abbreviations 37

Bibliography 38

1 Introduction

Galaxy [2] is a web-based, open-source scientific analysis platform designed to analyze large biomedical datasets commonly found in the fields of genomics, proteomics, metabolomics, and imaging [14]. For this, researchers can access more than 5500 tools provided by the Galaxy ToolShed [5]. Galaxy's graphical web interface enables even users without programming background to use sophisticated and powerful tools for various tasks in the field of bioinformatics in a visual and interactive way.

Currently, Galaxy allocates and provides a fixed amount of memory for each tool. This can result in over- or under-allocation of memory. As a consequence, jobs run with a particular tool may fail because there is not sufficient Random-Access Memory (RAM) for its usage, or resources may be wasted as the actual memory usage is much less than the space allocated by Galaxy.

The amount of Central Proccessing Unit (CPU), RAM, and processing time a tool takes to finish is dependent on the input dataset size and the computational complexity of the tool. By examining data which got recorded on the European instance of Galaxy [3], the goal of this work is to evaluate different Machine Learning (ML) methods to predict memory usage of tools based on the size of the input data and other features such as number of CPU cores. E.g. the tool *RNA-Star* uses in one job run 30 GB of memory and in a different run 10 GB. Suppose 20 GB of memory are allocated for this tool, then in one instance the job fails because of insufficient memory and in the other case too much would be assigned. Therefore, ML methods might be helpful to optimize this since they are able to automatically learn patterns and decision rules from data to make predictions on new unseen data points. In this case, the ML models learn to identify patterns and to predict memory usage by using the recorded *Filesize*, *Number of files* and *Slots* of jobs with the corresponding *Memory bytes* value.

1 Introduction

This report first discusses works that had similar approaches and contents (chapter 2) after which some theoretical basics of ML methods (chapter 3) are explained. Then, the concrete methods and applications of the dataset and models are described (chapter 4), which are further evaluated and assessed in chapter 5. Finally the findings of the project are discussed and future research possibilities are brought forward in chapter 6.

2 Related works

In the past, there have been several approaches aimed at optimizing resource allocation for services. The goal is the same as in this project: avoid wasting resources caused by the assignment of too much resources and prevent service failures when too less resources are allocated. Some of these methods are presented and compared to the work of the project in this chapter.

One similar approach was given by Schneider et al [34]. They trained ML models on Virtual Network Function (VNF) data. VNFs are virtualized network services that run on open computing platforms and were previously run on dedicated hardware [12]. Typical VNFs that are commonly used are virtualized routers, firewalls and Network Address Translation (NAT) services. The majority of VNFs are run on virtual machines (VMs) [12]. Normally, a VNF is invoked using a certain quantity of resources, which is set by the VNF developer beforehand. Nevertheless, VNFs usually do not require a constant amount of resources, but rather need more or less resources depending on the traffic load. A fixed allocation of resources can lead to under-allocation or over-allocation, meaning that VNFs are not able to handle all packets, resulting in packet loss and a reduction in quality of service [34]. For this reason, under-allocation is prevented by assigning a high amount of resources to each VNF instance. This on the other hand leads to waste of resources. These procedures are similar to the way Galaxy handles its resource allocation for jobs. The models of their work were trained such that they predict the resource requirements given a certain traffic load. Likewise to this work Schneider et al. compared various ML algorithms against each other including Linear Regressors, Support Vector Regression (SVR), Ensemble-based methods & Neural Networks. With their approach they were able to decrease the resource consumption up to 12 times lower compared to using standard fixed resource allocation [34].

Similar to the work by Schneider et al. [34], Nawrocki & Osypanka introduced a method that attempts to predict cloud resources [29]. By using anomaly detection methods and ML models to build knowledge, they were able to predict resource allocation, while taking into account various QoS (Quality of Service) constraints. With this they were able to reduce the costs ranging from 51% up to 85% (for PaaS/IaaS).

Ricart's work can also be classified under the same research area [32]. In his work, a method for resource prediction in the field of nanoelectric circuit design was produced. By using data from previous circuit designs he was able to predict resources, such as runtime, CPU time, RAM, leakage and area utilization. The prediction accuracy was compared with different ML model such as classification and regression trees (CART), linear models, SVR, MARS (Multivariate Adaptive Regression Splines), neural networks & Random Forests.

All presented works propose similar methods, namely the usage of ML models to predict computational or other forms of resources for different kinds of services. In this context, this project likewise brings forward a comparison of different ML algorithms, which aim to predict memory usage of jobs run on Galaxy Europe. The significant achievement of this work is the application of ML methods on data from the European Galaxy server [3], and in doing so, examines the execution of tools in the field of Deoxyribonucleic Acid (DNA)/Ribonucleic Acid (RNA) analysis and other areas of bioinformatics. One of the main contributions of this work to the task of resource prediction is that it gives insights into the resource consumption of tools run on Galaxy. In addition to that, this project computes uncertainties around predictions to allow investigations on the trade-off of resource costs and quality of service as described in Section 4.4.

3 Background

3.1 Random Forest

Random Forest is a commonly used supervised ML algorithm for classification and regression problems. It was first introduced properly by Breiman in 2001 [19]. A Random Forest is categorized as an ensemble learning algorithm, since it is constructed of multiple decision trees (see Figure 3.1).

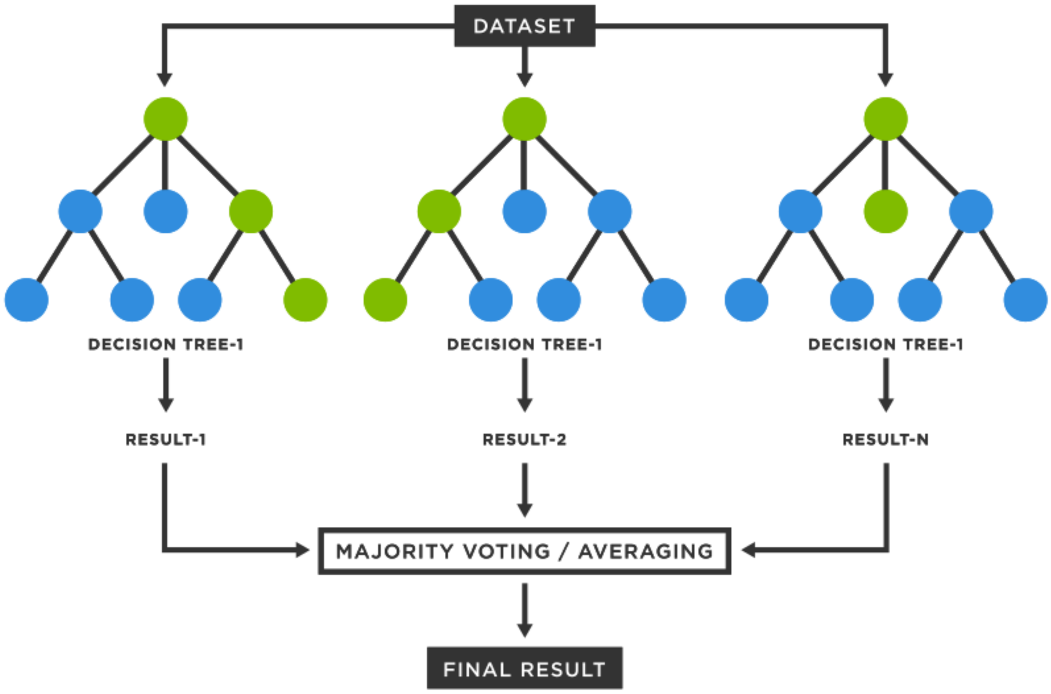


Figure 3.1: Structure of a Random Forest consisting of multiple decision trees [11]

A decision tree learns to predict a target feature by inferring simple decision rules from the dataset. It is comprised by a series of decision nodes, branches and leaves. Each decision node in the tree questions the data and splits it into two branches aiming to maximize the information gain by doing so [31]. A prediction can be done by starting at the root node and following the decisions until a leaf node is reached, which gives the output of the tree. An example of such a tree can be seen in Figure 3.2.

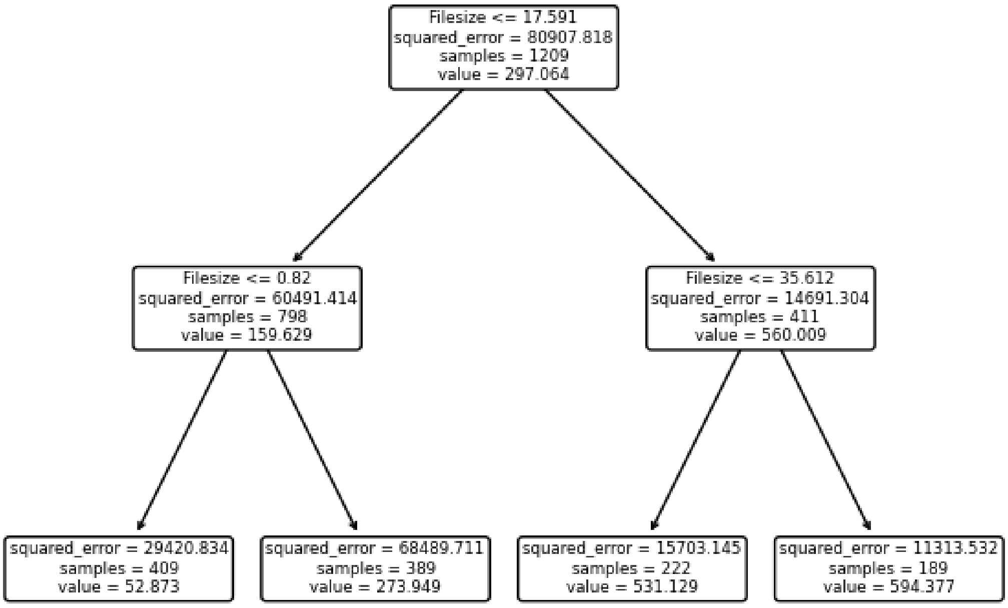


Figure 3.2: An example of a decision tree on the tool-resource-prediction dataset

The algorithm uses two types of randomness. On the one hand, it makes use of bagging, i.e. each decision tree selects random samples of the training dataset with replacement [26]. A single decision tree may overfit on data, limiting its capacity for generalization to unknown data [15]. Using multiple decision trees with different training sets, can lead to overall better generalization and reduce the variance significantly without increasing the bias [23]. The other source of randomness comes from randomly selecting a small sub-set of features for each decision tree. This de-correlates the decision trees even more, which in turn leads to better prediction of the ensemble [26]. For classification, the outcomes of the different decision trees are considered in a majority vote, while in the case of regression the mean of the trees is taken as prediction (see Figure 3.1).

3.2 Extreme Gradient Boosting

XGBoost (Extreme Gradient Boosting) is a library that provides ML algorithms on the basis of gradient boosting algorithms [13]. The first regression gradient boosting algorithms were introduced by Friedman [22]. The model that XGBoost uses is basically constructed out of the same building blocks as a Random Forest (see Section 3.1), i.e. the model is made up of an ensemble of decision tress. The main difference comes from how each model is trained during the learning process. XGBoost uses a concept called "boosting" where weak models are combined into a single strong model in an iterative way, which helps to reduce the bias and variance [18]. So instead of combining all the decision trees parallelly like in Random Forest, the trees are combined sequentially. The idea is that by using multiple trees after each other, the current tree corrects the error of the previous tree or in other words, each tree "boosts" the attributes that led to errors or misclassifications of the previous tree [27]. The structure of the Extreme Gradient Boosting (XGB) model can be seen in Figure 3.3.

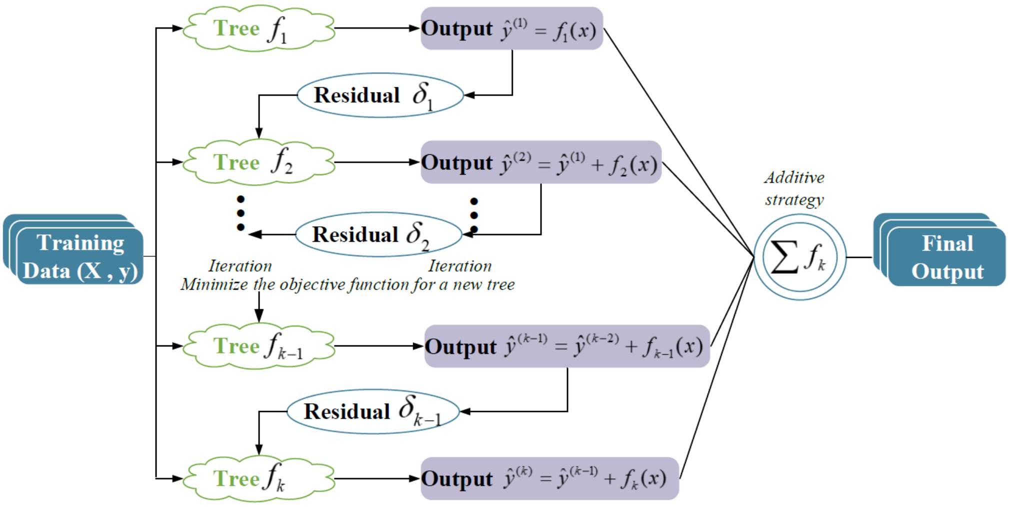

Tree f_1
Output $\hat{y}^{(1)} = f_1(x)$
Residual δ_1
Tree f_2
Output $\hat{y}^{(2)} = \hat{y}^{(1)} + f_2(x)$
Residual δ_2
Training Data (X , y)
Iteration
Iteration
Minimize the objective function for a new tree
Additive strategy
$\sum f_k$
Final Output
Tree f_{k-1}
Output $\hat{y}^{(k-1)} = \hat{y}^{(k-2)} + f_{k-1}(x)$
Residual δ_{k-1}
Tree f_k
Output $\hat{y}^{(k)} = \hat{y}^{(k-1)} + f_k(x)$

Figure 3.3: Structure of the XGB model [21]

The output of the model is calculated with K additive functions expressed by the following equation [20]:

$$\hat{y}_i = \phi(\boldsymbol{x_i}) = \sum_{k=1}^{K} f_k(\boldsymbol{x_i}), f_k \in \mathcal{F}, \tag{3.1}$$

where $\mathcal{F}$ is the space of the decision trees, f_k represents an individual decision tree, $\boldsymbol{x_i}$ is sample i of the dataset and $\hat{y_i}$ is the output for sample i.

The model aims to minimize the following objective [20]:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y_i}, y_i) + \sum_k \Omega(f_k), \tag{3.2}$$

where l is a differentiable convex loss function measuring the difference between the prediction $\hat{y_i}$ and the target y_i. Ω is a regularization term. The regularization term penalizes the complexity of the model (i.e. the complexity of a single decision tree), which leads to better generalization and less overfitting.

Since the model is trained in an additive and sequential way, following objective is minimized at the t-th iteration with the i-th instance [20]:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\boldsymbol{x_i})) + \Omega(f_t) \tag{3.3}$$

This implies that the f_t (so the decision tree) that improves the model the most, according to equation 3.2, is added greedily [20].

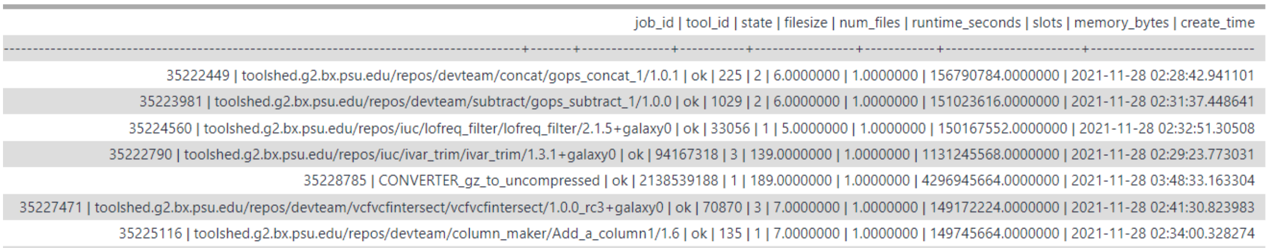
4 Methods

This chapter explains the technical specifications of the methods used in the project. First, the Galaxy data set and the data preprocessing steps are described, followed by a description of the training, testing, and hyperparameter optimization steps. In the final section 4.4 a method to do predictions with uncertainty is brought forward.

4.1 Galaxy job run dataset

4.1.1 Description of the dataset

The Galaxy server collects extensive data on the jobs run on the platform. This data can be used and analyzed to improve the resource allocation, such that the assigned memory for each job can be optimized. The Galaxy job run dataset used for this project has been collected for the period from 31.12.2019 to 23.05.2022 on the European server of the Galaxy network. In total the dataset consists of approximately 38 million entries with about 10 GB in total size. The Galaxy job run dataset can be found here. In Table 4.1 the recorded job attributes are shown. An excerpt of the dataset can be seen in Figure 4.1.



```
job_id | tool_id | state | filesize | num_files | runtime_seconds | slots | memory_bytes | create_time
-------------------------------------------------------------------------------------+-------+--------------+----------+----------------+-----------+----------------------+--------------------------
35222449 | toolshed.g2.bx.psu.edu/repos/devteam/concat/gops_concat_1/1.0.1 | ok | 225 | 2 | 6.0000000 | 1.0000000 | 156790784.0000000 | 2021-11-28 02:28:42.941101
35223981 | toolshed.g2.bx.psu.edu/repos/devteam/subtract/gops_subtract_1/1.0.0 | ok | 1029 | 2 | 6.0000000 | 1.0000000 | 151023616.0000000 | 2021-11-28 02:31:37.448641
35224560 | toolshed.g2.bx.psu.edu/repos/iuc/lofreq_filter/lofreq_filter/2.1.5+galaxy0 | ok | 33056 | 1 | 5.0000000 | 1.0000000 | 150167552.0000000 | 2021-11-28 02:32:51.30508
35222790 | toolshed.g2.bx.psu.edu/repos/iuc/ivar_trim/ivar_trim/1.3.1+galaxy0 | ok | 94167318 | 3 | 139.0000000 | 1.0000000 | 1131245568.0000000 | 2021-11-28 02:29:23.773031
35228785 | CONVERTER_gz_to_uncompressed | ok | 2138539188 | 1 | 189.0000000 | 1.0000000 | 4296945664.0000000 | 2021-11-28 03:48:33.163304
35227471 | toolshed.g2.bx.psu.edu/repos/devteam/vcfvcfintersect/vcfvcfintersect/1.0.0_rc3+galaxy0 | ok | 70870 | 3 | 7.0000000 | 1.0000000 | 149172224.0000000 | 2021-11-28 02:41:30.823983
35225116 | toolshed.g2.bx.psu.edu/repos/devteam/column_maker/Add_a_column1/1.6 | ok | 135 | 1 | 7.0000000 | 1.0000000 | 149745664.0000000 | 2021-11-28 02:34:00.328274
```

Figure 4.1: Excerpt of the Galaxy job run dataset

Attribute	Description
job_id	Unique id representing each job
tool_id	The name of the tool (with version) used for the job
state	Flag that indicated if job was run successfully
filesize	Total size of the files used for the tool in bytes
num_files	The number of files used for the job
runtime_seconds	Total runtime of the job in seconds
slots	Number of CPU cores assigned by Galaxy
memory_bytes	Maximum memory usage in bytes
create_time	Creation date and time of the job

Table 4.1: The recorded job attributes

To reduce the dataset size and make it more manageable, additional preprocessing steps have been done, which are described in the following section.

4.1.2 Preprocessing

Reducing dataset size

For the project's approach only a subset of the recorded attributes of the original Galaxy job run dataset are relevant. The attributes *job_id & runtime_seconds* can be removed, since they are not necessary for this work. The attribute *state* can be left out, since the query to fetch the data from the Galaxy server already only considered jobs, which were run successfully (i.e. *state* == *OK*). The remaining relevant attributes are listed in Table 4.2.

In addition to this, empty characters as well as trailing zeroes for the attributes *Filesize, Number of files, Slots & Memory bytes* were removed, since these fields are integers and do not need decimal places. Furthermore, some entries contained no values for *Filesize, Number of files or Memory bytes* or had invalid combinations for *Filesize & Number of files* (e.g. *Number of files* = 0 but *Filesize* > 0) which were also removed.

These steps reduced the total size of the dataset to about 3 GB with about 26 million entries. In this process approximately 12 million records were discarded.

Relevant attributes
tool_id
filesize
num_files
slots
memory_bytes
create_time

Table 4.2: Relevant job attributes

Filtering faulty data

Currently the tools run on the Galaxy platform are assigned a fixed amount of memory. How much each tool is assigned is given in the Galaxy repository by the tool destinations file [10]. Tools which are not configured in this file are assigned a default value of 1 GB for memory. The upper bound of any Galaxy tool is 1 TB. If a job fails, it is restarted with twice the amount of memory as before. Tools can get restarted up to a maximum of 3 times. This means that that the maximum memory usage (if the job gets restarted 3 times) is given by:

$$max\_memory = initial\_assigned\_memory * 2^3 = initial\_assigned\_memory * 8 \quad (4.1)$$

Using this information, the dataset was filtered for entries, which exceed Memory bytes over 1 TB as well as entries, that surpass the maximum value given by Equation 4.1. This further reduced the dataset to about 2.75 GB. These removed faulty entries were collected into a separate file which can be found here. This might be useful in the future if one decides to investigate possible errors in the data recording step of Galaxy.

Regardless of whether such errors are due to any bugs in the code, server malfunction, errors in the recording process, or a mix of any of these things, the fact that these errors exist puts the validity of the rest of the dataset in question.

Removing outliers

Through data analysis, it occurred that some tools have data points whose *Memory bytes* are far away from most other data points. Since it is not possible to find out if jobs in the dataset are initial runs or whether they got repeated, it is assumed that these outliers are repeated jobs. Since the goal is to predict the memory of jobs in the initial run, such data points should be removed. This is done by calculating the mean and the standard deviation of *Memory bytes* and remove points that lie above the mean plus 2 times the standard deviation. To understand the influence of these outliers, experiments have been done which are discussed in Section 5.2.

4.2 Training and evaluation pipeline

This section of the report outlines several ideas related to the training and evaluation of the ML models. For the comparison of different methods Random Forest, XGB, Linear Regression and SVR were selected as ML models. The implementation of these models can be found in the *estimator.py* file which lies in the *src* folder of the master project GitHub repository [17].

4.2.1 Base estimators

Base estimators are the model instances which are used to achieve baseline performance. For the base estimators of the Linear Regression and SVR all parameters were kept as their default. The same was done for Random Forest and XGB with the difference of setting the number of trees, *n_estimators*, to 200 & *random_state* to 0 for both models. The hyperparameters of all models are later optimized as described in Section 4.3 and shown in Section 5.5.

4.2.2 Training and evaluating the estimators

For the training of the estimators the features *Filesize*, *Number of files* and *Slots* are used as input and *Memory bytes* is used as the target. The training and evaluation process in general is structured as following: A specific version of a tool is used as data and it is loaded. After this the data points are split into train and test set with a proportion of 80% and 20% respectively. The train set is then used to fit the model using 5-fold cross-validation after which the best model is extracted and evaluated on the test set. Cross-validation is helpful since it helps to avoid overfitting and it gives us initial insights about which methods, procedures and measures work best. The performances of the models are hereby evaluated using the R2 Score which is given in Equation 4.2, where y_i is the target of a sample, $\hat{y}_i$ is the prediction for this sample and $\bar{y}$ is the mean of the targets [1]. The best possible score is 1.0, and it can be negative as the model can be arbitrarily bad. In the common case where the target y is not constant, a model that always predicts the mean of y, ignoring the input features, would receive a score of 0.0 [7]. The R2 Score expresses the proportion of the variance of y explained by the independent variables of the model. This gives an indication of the quality of a fit and is thus a measure of how well unobserved samples are likely to be predicted by the model [7].

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{4.2}$$

4.3 Hyperparameter Optimization

After finding out which methods and approaches work best, Hyperparameter Optimization is applied using *HalvingGridSearchCV* with a 5-fold cross-validation [8]. *HalvingGridSearchCV* in contrast to normal *GridSearchCV* implements the *successive halving* mechanism [6]. *Successive halving* is an iterative selection process that, in the first iteration, evaluates all parameter combinations with a small number of resources. In each iteration only a subset of the candidates survive, while the resources are increased. In the end, only the candidate that is consistently among the top candidates across all iterations remains. This mechanism is of great use since not all possible parameter combinations are evaluated e.g. with full sample size. Instead, a proportion of the resources is used which reduces the time of the optimization process [6]. The results of this optimization are described in Section 5.5. The following parameter spaces are explored during the optimization process:

Random Forest parameter space:

- n_estimators = [50, 100, 200, 500]. Represents the number of decision trees.

- max_depth = [None, 4, 16, 32]. This is used to avoid overfitting. Experiments showed that most of the time the depth of the trees varies between 2 and 45, so these values are covered. With *None* the tree is expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

- min_samples_split = [2, 4, 8]. Increasing this value reduces the number of splits, therefore avoiding overfitting.

- min_samples_leaf = [2, 4, 8]. Helps to control overfitting as well.

XGB parameter space:

- learning_rate = 8 samples from log_space [0.003 to 0.3]

- n_estimators = [50, 100, 200, 500]

- max_depth = [2, 6, 16, 32, 64]. 6 is default tree depth for XGB. This parameter can avoid overfitting.

The Linear Regressor is not optimized since the only available hyperparameters for this are *fit_intercept, normalize, n_jobs, positive* and *copy_X* neither of which influences the learning process. *Normalize* could be used, but since *StandardScaler* is used before fitting the model, this is redundant.

SVR parameter space:

- kernel = ["rbf", "sigmoid", "poly"]

- C = [0.01, 0.1, 0.5, 1, 2, 4]

- gamma = ["scale", 0.001, 0.01, 0.1, 1]

4.4 Prediction with uncertainty

As discussed previously, it can happen that jobs have to be restarted for various reasons. One possible reason is that the initially allocated memory is not sufficient for the run. If a job fails, it will be restarted with twice as much memory as before (as mentioned earlier in Section 4.1.2). One concern therefore is to investigate whether it is possible to give the algorithms the probability with which a job should succeed on the first attempt. This gives us the opportunity to trade off between optimizing resources as much as possible and the ability of a tool to succeed on the first try.

To elaborate, an example is discussed: the algorithm is told that for the execution of a tool the job should be successful on the first attempt with a probability of 70% in terms of the required memory. For this instance, the algorithm predicts that the job will require 10 GB of memory. In another case, the algorithm is told that the job should be executed successfully with a higher probability e.g. with 90% on the first try. From this constraint, the algorithm in turn derives a prediction of 15 GB. This illustrates the option to trade off between a better resource allocation and the successful execution of a job.

4.4.1 Prediction intervals

The implementation of this functionality is possible for the Random Forest but not for the XGB model using prediction intervals. A prediction interval specifies an interval in which the prediction is expected to be with a particular probability [24]. It provides probabilistic upper and lower bounds on the estimate of the algorithm. This is useful because it is often advantageous to specify an interval with a certain degree of confidence, that contains the response value, rather than giving a single number as the predicted value. For example a 95% prediction interval for the new data point x and the prediction y is given by [28]:

$$I(x) = [Q.025(x), Q.975(x)] \tag{4.3}$$

This implies that the prediction y lies with high likelihood in the interval $I(x)$. The range of this prediction interval can differ significantly with x and depends on the model's training data [28].

In Figure 4.2 an example can be seen where about 90% of the samples were to lie within the prediction intervals, and in Figure 4.3 the percentage for the prediction intervals was set to 50%. In both figures, it can be observed that the prediction intervals are smaller for more

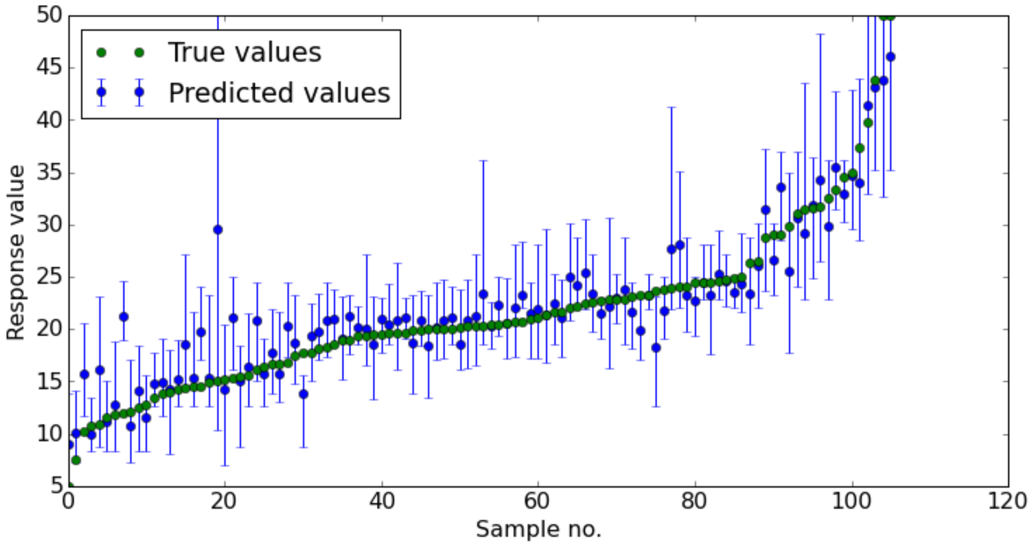


Figure 4.2: Plot of the true values and 90% prediction intervals [33]

accurate predictions due to the fact that the model has more confidence in the prediction at these points.

Quantile Regression Forests are one way to determine confidence intervals for Random Forests [28]. The underlying concept of Quantile Regression Forests is that rather than recording the average value of the response variable in each tree leaf in the forest, all observed responses in the leaf are recorded [33]. Thereby the full conditional distribution $P(Y <= y|X = x)$ of the response values for each x can be obtained. As an example, the 90% prediction intervals would comprise the area between the 5 and 95 percentiles of the distribution of the response variables.

To get the prediction intervals from a Random Forest, all of the individual decision trees have to be expanded fully, such that each leaf contains only one value [33]. Then, a prediction gives us the individual response variables on which the distribution is built on, if the forest is sufficiently big.

To use the prediction intervals with the algorithm, the following steps are performed: The user gives as input a dataset of a particular tool on which to make the prediction and a probability between [0.0, 1.0], which is used to calculate the prediction intervals. The algorithm then

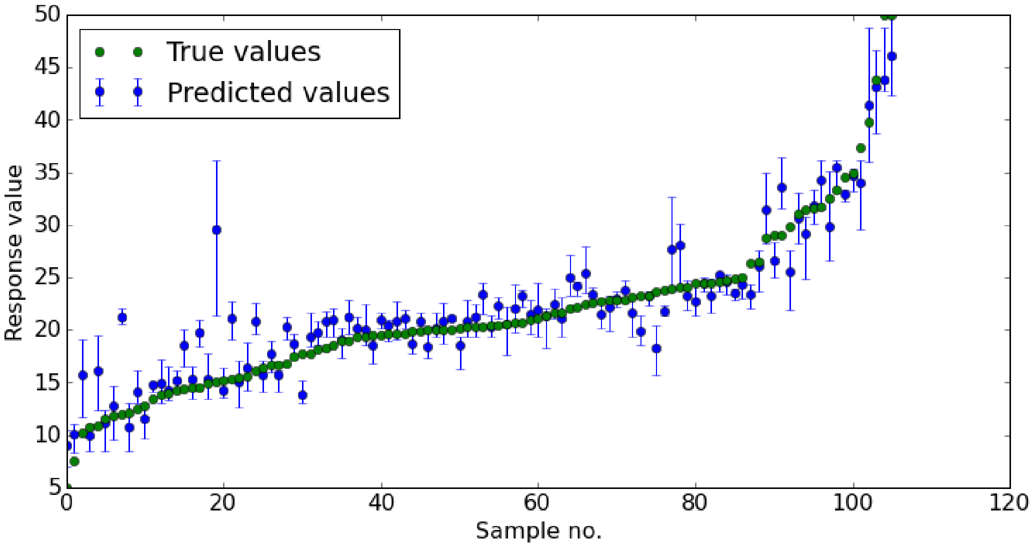


Figure 4.3: Plot of the true values and 50% prediction intervals [33]

takes the maximum value of each prediction interval as the prediction. In theory, a job run on Galaxy with this prediction is less likely to fail due to lack of memory.

5 Results

5.1 Removing faulty data

First of all, experiments were done to ascertain the effects of the data pre-processing steps. In this part the removal of faulty data as described in Section 4.1.2 was inspected. To do so, models were trained and evaluated on 5 different tools. The 5 tools were selected because they had frequent faulty entries as was found during pre-processing. In total about 2.5% of the Galaxy job run dataset turned out to be faulty. About 6.5% of *fastqc/0.72* were faulty. For *ivar_trim/1.2.2*, *ivar_removereads/1.2.2*, *cutadapt/1.16.5* and *mimodd_reheader/0.1.8_1* the proportions of faulty amounted to 2%, 2.1%, 22% and 1.5% respectively. The models were trained using the parameters given in Section 4.2.1 and the pipeline described in Section 4.2.2. In order to allow a fair comparison, each individual dataset consists of 10000 samples. Out of these 10000 samples, 500 samples represent faulty data as defined in Section 4.1.2. In Table 5.1 the R2 scores, before and after removing faulty data, are shown for each model. As Table 5.1 shows, almost all models display an improvement across the different datasets when the faulty data is removed. Although the datasets are comprised of only 5% faulty data, the improvements are very drastic. There is only one case where the R2 score decreased, and this is for dataset *mimodd_reheader/0.1.8_1* & model *XGB*. A look at the training results shows that the mean absolute error has improved from 10.30 → 0.03 (since the R2 score is more meaningful to

Dataset	Random Forest	XGB	Linear Regression	SVR
fastqc/0.72	-0.13 → 0.85	-0.17 → 0.86	0.00 → 0.82	-0.01 → 0.89
ivar_trim/1.2.2	-0.93 → 0.86	-1.50 → 0.87	0.00 → 0.77	-0.02 → 0.84
ivar_removereads/1.2.2	-0.51 → 0.64	-0.64 → 0.68	0.00 → 0.74	-0.03 → 0.70
cutadapt/1.16.5	-0.82 → 0.89	-1.64 → 0.89	0.05 → 0.13	0.05 → 0.90
mimodd_reheader/0.1.8_1	-0.37 → -0.12	-0.47 → -0.51	0.00 → 0.37	-0.02 → 0.16

Table 5.1: R2 Score of the models before and after removing faulty data

interpret predictions, the mean absolute error is not used in this work). This indicates that there are a few (perhaps only one) bad predictions that are strongly penalized by the R2 function, while the mean absolute error does not penalize this nearly as much.

5.2 Removing outliers

Another pre-processing step that needs to be examined is the removal of outliers as explained in Section 4.1.2. For this 4 of the most popular tools were selected with 10000 samples per tools. The models were trained using the parameters given in Section 4.2.1 and the pipeline described in Section 4.2.2. As can be seen in Table 5.2 especially the Random Forest and XGB models benefit from the removal of the outliers. E.g. for *rna_star/2.7.5b* the R2 Score of Random Forest improves from -0.33 to 0.04 which is very significant. The same can be said about the XGB model. The Linear Regressor and SVR do not benefit from the outlier removal as much as the other models. In some cases it even worsens the scores.

Dataset	**Random Forest**	**XGB**	**Linear Regression**	**SVR**
rna_star/2.7.5b	-0.33 → 0.04	-0.35 → 0.07	0.05 → 0.03	0.06 → 0.06
bowtie2/2.3.4.3	-0.18 → 0.07	-0.15 → 0.07	0.04 → 0.01	0.05 → 0.05
hisat2/2.1.0	-0.15 → 0.06	-0.05 → 0.07	0.10 → 0.06	0.06 → 0.05
bwa_mem/0.7.17.1	-0.21 → -0.01	-0.28 → 0.00	0.01 → -0.01	0.00 → -0.01

Table 5.2: R2 Score of the models before and after removing outliers

5.3 Correlation analysis

To get a better understanding of the underlying relations of the dataset, the Pearson correlation coefficient between *Filesize* and *Memory bytes* was calculated for each tool. The results of these calculations were saved to a separate file [16]. Additionally, the number of samples per tool in the dataset was saved. This is helpful because many tools have a perfect correlation of 1.0, due to the fact that there are only few samples (often only 2).

One thing that stands out is that some of the tools (about 300 of total 4800 tools) have a moderate or even strong negative correlation. While this is nothing inherently wrong, one

Version	Pearson correlation Filesize ⟷ Memory bytes	nr_samples
0.4	0.99	5
2.3.4.1	0.74	130
2.3.2.2	0.53	37
2.4.2	0.31	48286
2.4.5	0.30	6031
2.3.4.3	0.25	97465
2.2.6.2	0.24	161
2.3.4.2	0.21	355
0.2	0.01	16
0.3	-0.16	3

Table 5.3: Pearson correlation for different versions of *bowtie2*

would assume that as the file size increases, the memory consumption would also go up, so a positive correlation should be present. As mentioned earlier in other sections, this makes the validity of the dataset put into question. Another dubious aspect is that in some cases there are major differences in the correlation between different versions of the same tool. As can be seen in Table 5.3. Of course, this must be taken cautiously, as the number of samples is not equal for all versions to make a fair comparison.

Using the information from the calculated Pearson correlation coefficients, two types of experiments were conducted, which are described in the following sections.

5.3.1 Moderate-high correlation

In this part, the ability of the models to make predictions based on data sets that show moderate to strong correlation is examined. For this, tools with moderate correlations of 0.35 up to strong correlations of 0.95 were used, which are listed in Table 5.4. For each tool 10000 samples were taken as input and the models were trained & evaluated using 5-fold cross validation. The results of the runs can be seen in Figure 5.1.

Figure 5.1 clearly shows that for datasets with very high correlation, all models perform well. For the tool *lofreq_indelqual/2.1.4* Random Forest, XGB and Linear Regression perform equally well, having a R2 Score of about 0.95 or higher. For this case the Support Vector

Dataset	Pearson correlation coefficient
lofreq_indelqual/2.1.4	0.95
bamleftalign/1.3.1	0.74
fastqc/0.73	0.66
umi_tools_extract/0.5.5.1	0.55
bedtools_intersectbed/2.30.0	0.35

Table 5.4: Tools with moderate to high correlation

Regressor performs the worst. This could be due to the fact that this model has more hyperparameters than the other models to choose from, which may need to be optimized.

For the tools *bamleftalign/1.3.1* and *fastqc/0.73*, which have a lower correlation, the Linear Regressor performs worse since there is no longer a clear pattern of linearity visible in the data. The other models perform all equally well with R2 Scores of about 0.75 for *bamleftalign/1.3.1* and 0.9 for *fastqc/0.73*. As for the data with even lower correlation, all models do not perform as well with an R2 score varying between 0.0 and 0.2.

Overall, a clear tendency can be observed, which is that, the greater the correlation between *Filesize* and *memor_bytes*, the better all models perform. This trend can also be seen in Figure 5.2 where the model performances are plotted against the correlation of each dataset.

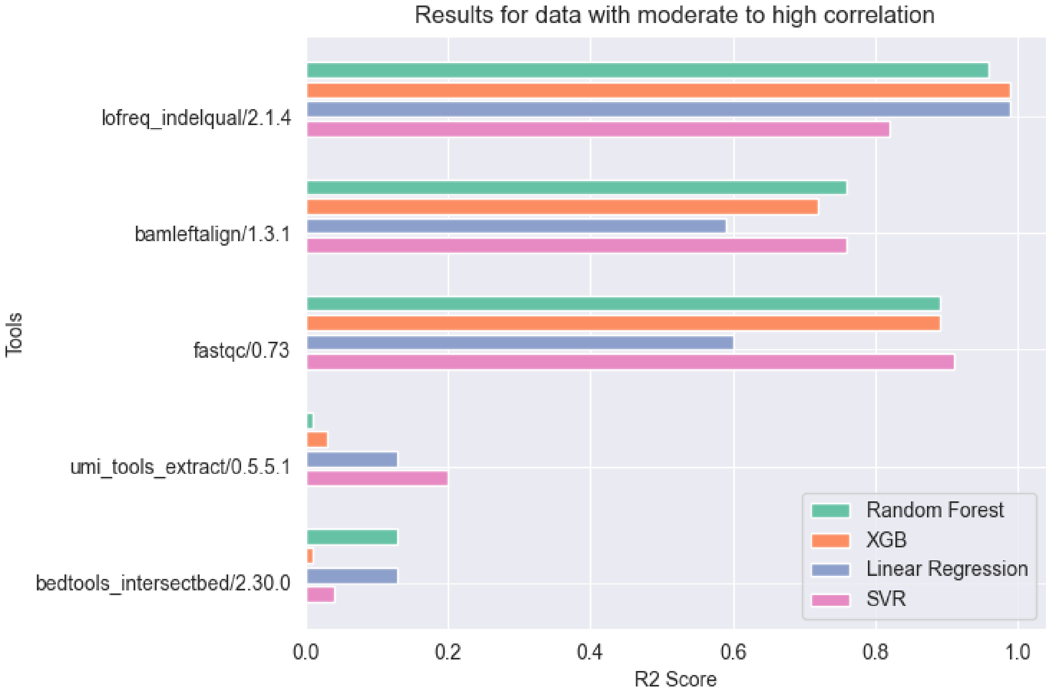


Figure 5.1: R2 Score of the models using moderate-high correlation data

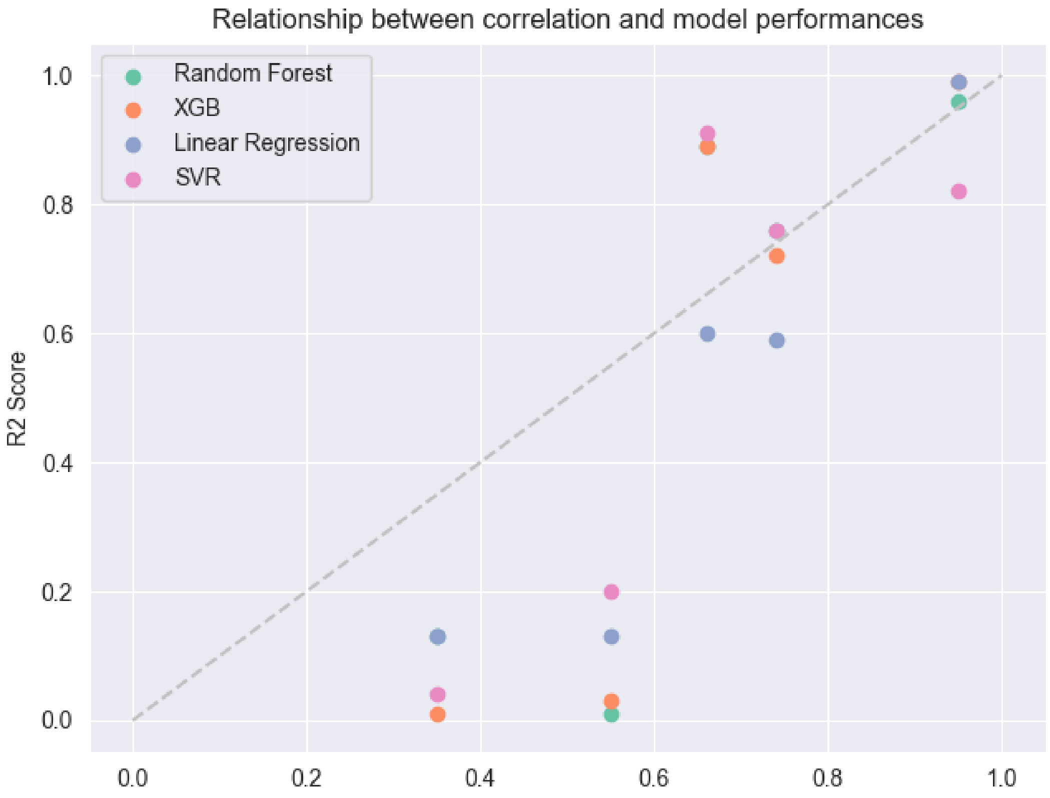


Figure 5.2: Model performance in relation to correlation for moderate-high correlation data

5.3.2 Low-moderate correlation

The same examination as in Section 5.3.1 can be done for tools which show low to moderate Pearson correlation between *Filesize* and *Memory bytes*. A subset of 5 tools that meet this criteria are listed in Table 5.5. Their correlation coefficients range from 0.01 up to 0.45.

Dataset	Pearson correlation coefficient
fastp/0.20.1	0.45
vcf2tsv/1.0.0_rc3	0.31
rna_star/2.7.2b	0.16
bg_diamond/2.0.8.0	0.07
samtools_idxstats/2.0.3	0.01

Table 5.5: Tools with low to moderate correlation

Using the set from Table 5.5 with 10000 samples per tool, all models have been trained applying 5-fold cross-validation, after which the best instance has been evaluated on the test set. The results of these runs can be seen in Figure 5.3.

For the dataset with the highest correlation of 0.45 *fastp/0.20.1*, all models beside the Linear Regressor perform well, with an equally good R2 Score of around 0.6. For the other datasets which have a correlation of 0.31 or lower, the R2 Score of all models vary greatly, with the general performance not being that good. What stands out is that the XGB model performs the worst for all datasets beside *fastp/0.20.1*, with R2 scores ranging from -0.6 to -0.15. This again might be due to the fact that the hyperparameters of XGB need to be tuned more than for the other models.

In Figure 5.4 the model performances are plotted against the correlation of the datasets. Again, a clear upward trend in the performance of the models is evident with increasing correlation of the dataset. The only dataset that stands out is the tool *bg_diamond/2.0.8.0* which has a correlation of 0.07. For this tool the Random Forest, Linear Regressor and SVR perform relatively well compared to the data points around it. This might be explained by looking at the importance the Random Forest model gives to each input feature. These are given in Table 5.6. For all datasets beside *bg_diamond/2.0.8.0*, the feature *Filesize* was given an importance of 1.0. For *bg_diamond/2.0.8.0* the importance of *Filesize* and *Number* *of files* is almost equally important with their corresponding values being 0.55 and 0.45. This is probably the explanation why for *bg_diamond/2.0.8.0* the models partly perform better.

Table 5.6 moreover illustrates one of the major drawbacks of the dataset: the models rely mostly on the *filesize* feature, which means when there is no greater correlation between *Filesize* and *Memory bytes*, the models perform bad or worse. The models work best when there is a high correlation between Filesize and Memory bytes as shown in Section 5.3.1. In addition to that the input feature *Slots*, which resembles the number of CPU Cores assigned by Galaxy for the job, is equal nearly in every sample in each dataset. Therefore no useful information can be extracted and accordingly the feature importance is 0.0. All in all, there are too few useful features in the dataset, so the models are too dependent on only one feature. This is even more problematic since about 50% of the tools show a correlation between *Filesize* and *Memory bytes* lower than 0.5, meaning that prediction for a majority of the tools is difficult.

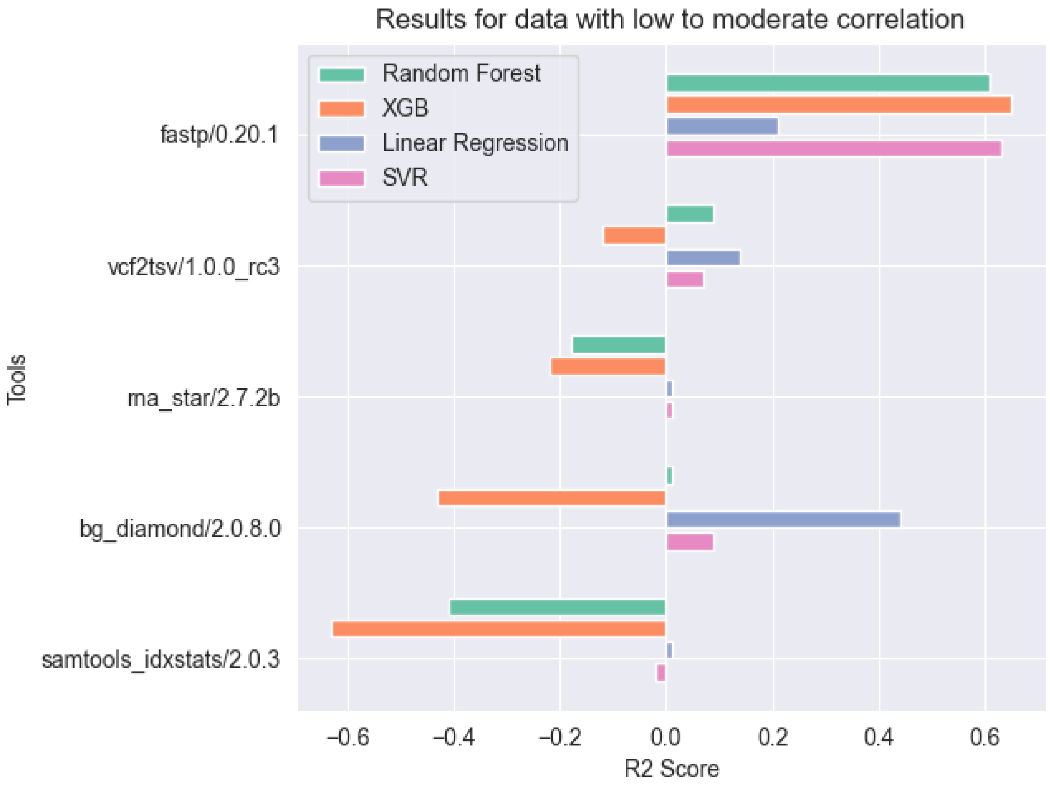


Figure 5.3: R2 Score of the models using low-moderate correlation data

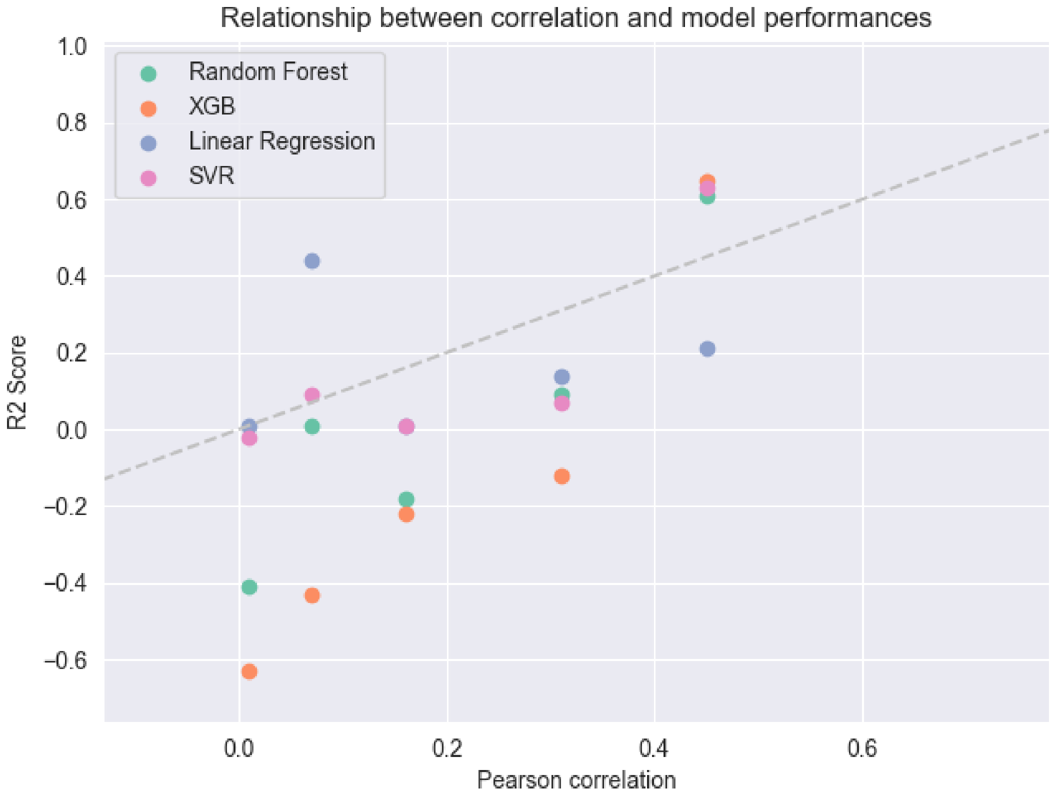


Figure 5.4: Model performance in relation to correlation for low-moderate correlation data

	Feature importance		
Dataset	Filesize	Number of files	Slots
fastp/0.20.1	0.97	0.03	0.00
vcf2tsv/1.0.0_rc3	1.00	0.00	0.00
rna_star/2.7.2b	0.97	0.03	0.00
bg_diamond/2.0.8.0	***0.55***	***0.45***	0.00
samtools_idxstats/2.0.3	1.00	0.00	0.00

Table 5.6: The given importance of Random Forest for the input features

5.4 Performance on other datasets

In addition to the data that was used from the European Galaxy instance, data from the main Galaxy server [4] was accessed, which was provided by Kamali [25]. Unfortunately, the data is limited to a total of only 2 tools, namely bowtie2 and hisat2, where the used version for both tools is not known. After transforming the data to a format usable for this work, the Pearson correlation between *Filesize* and *Memory bytes* was calculated with the following results:

Dataset	Pearson correlation	Number samples
hisat2 (Galaxy main)	0.66	2797
bowtie2 (Galaxy main)	0.62	3963

Table 5.7: Correlation of data from Galaxy Main [25]

As seen in Figures 5.5 and 5.6 both datasets show a logarithmic growth. However for *hisat2* there are some data points at the top of Figure 5.6 which show a strong variance in *Filesize* while having the same amount of *memory*. The same pattern is visible for *bowtie2* in Figure 5.5. In addition to that, on the left side of Figure 5.6 there are data points visible which have a big variance in *Memory bytes* while having a fixed *Filesize* around 0 GB. These patterns exist not only in the data from Galaxy main but also in the data used for this project as well. As mentioned more than once, they cast doubt on the data recording process of Galaxy. How should the execution of a job with a Filesize of a few kilobytes, consume 30 GB of memory, while Filesizes from 10 to 20 GB consume the same amount?

Using the data from both of these tools, all models were trained and evaluated as described in Section 4.2.2. The results can be seen in Figure 5.7. The Linear Regressor performed worst out of all the models for both tools. This might be due to the non-linear shape of both datasets as can be seen in Figures 5.5 and 5.6. For *bowtie2* all models performed equally well with an R2 Score of about 0.9. Regarding the hisat2 dataset, SVR scored worse than Random Forest and XGB with an R2 Score of approximately 0.55. In contrast Random Forest and XGB had about the same score of 0.7. All in all, the models performed worse for *hisat2* than for *bowtie2*. This can be caused by multiple reasons. One is that the dataset for *hisat2* has about 30% fewer samples than *bowtie2*. Therefore, this might effect the generalization performance. Another possible reason is that, as discussed and shown in

Figure 5.6, there exist a substantial amount of data points that show a great variance for fixed *Filesize* or *Memory* which possibly leads to a worse score.

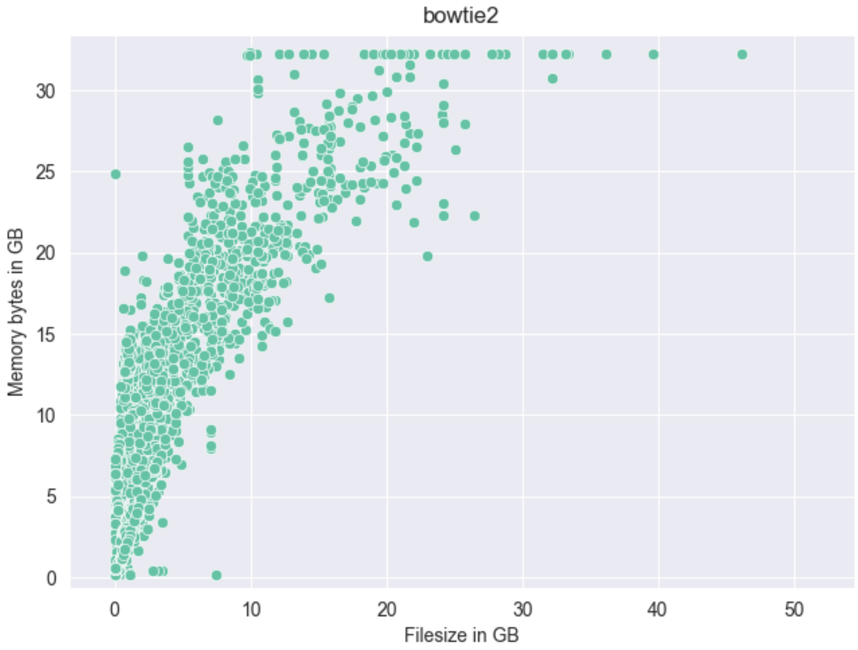


Figure 5.5: Filesize vs Memory bytes of bowtie2 (Galaxy main)

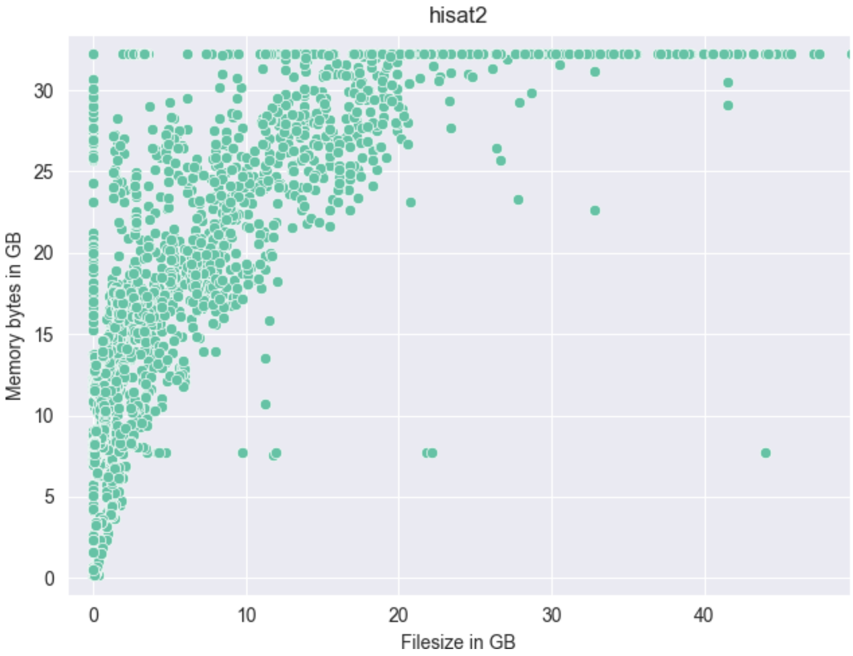


Figure 5.6: Filesize vs Memory bytes of hisat2 (Galaxy main)

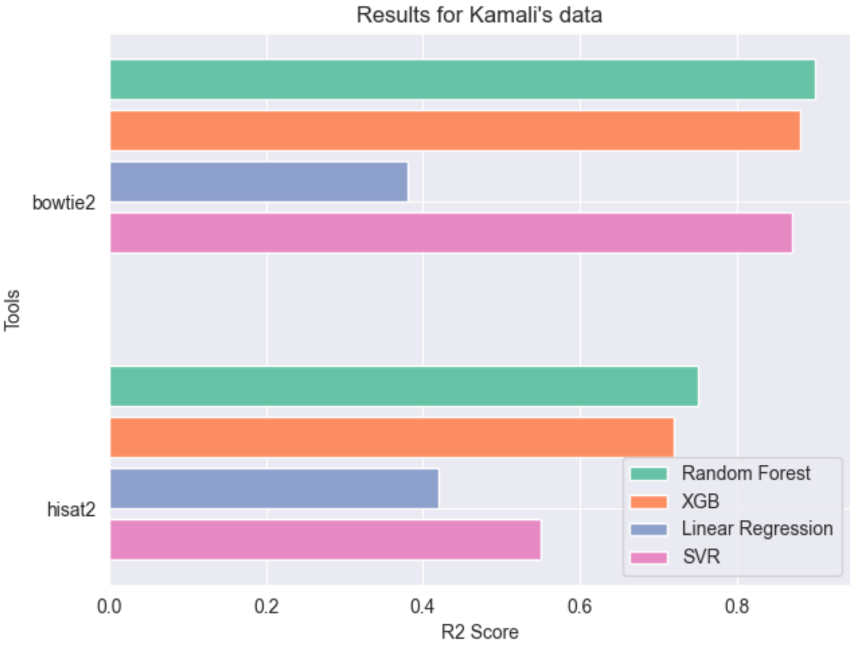


Figure 5.7: Results for Galaxy main's data

5.5 Hyperparameter Optimization & comparison to Galaxy baseline

In this section a comparison to Galaxy's current method of assigning fixed amounts of memory for the execution of tools is done.

To get a representative selection across all tools, 4 groups of tools were defined. For each group, two tools were selected with 10000 samples respectively. The selected tools are shown in Table 5.8 with their corresponding Pearson correlation and average memory usage.

Group	Tool	Pearson Correlation	Average Memory [GB]
High Correlation	lofreq_indelqual/2.1.4	0.95	0.21
	bamleftalign/1.3.1	0.74	0.34
Low Correlation	vcf2tsv/1.0.0_rc3	0.28	0.14
	rna_star/2.7.2b	0.20	56.72
High Memory	rna_star/2.7.5b	0.24	60.35
	kraken2/2.1.1	0.10	36.82
Low Memory	fasta2tab/1.1.1	0.29	0.20
	gmx_sim/2020.4	0.21	0.50

Table 5.8: Selected tools with their corresponding group, correlation and memory usage

To evaluate the best possible memory prediction, the models need to be optimized first. For this Hyperparameter Optimization is done as described in Section 4.3. In addition to that StandardScaler from the scikit-learn library was used for SVR and Linear Regression since both models are not scale invariant in contrast to Random Forest and XGB. StandardScaler removes the mean and scales the data to unit variance such that it has a normal distribution form. This is beneficial since SVR and Linear Regression behave worse if the features are not normally distributed [9]. In Table 5.9 the R2 Scores for each model & dataset is shown before and after doing Hyperparameter Optimization. For each row (dataset) the best model is highlighted.

In almost all cases across the models, there were improvements by optimizing the hyperparameters. For Random Forest and XGB the improvements were very significant. E.g. Random Forest had an R2 Score of -0.33 for *rna_star/2.7.5b* before and a score of 0.09 after optimization. The XGB model had about the same improvement for *rna_star/2.7.5b*. As expected all models perform well on *lofreq_indelqual/2.1.4* since it has a strong correlation of 0.95 between *Filesize* and *Memory bytes*. For all other datasets Random Forest was the model with the best score, beside for tool *fasta2tab/1.1.1* where SVR performed best. However, there was often not much of a difference between the Random Forest and XGB scores. For the SVR model the optimization has not improved the scores as much as hoped. Often the score got slightly worse, which may be caused by the successive halving technique used in the optimization process. In general the tools for which all models performed the best were *lofreq_indelqual/2.1.4*, *bamleftalign/1.3.1* and *fasta2tab/1.1.1*. The first two are plausible since they have a strong correlation. The tool *fasta2tab/1.1.1* on the other hand has a low correlation of 0.29. The good results for this tool can be explained by looking at Figure 5.8 where, beside a few outliers, a clear pattern of memory usage can be seen with increasing Filesize.

Dataset	Random Forest	XGB	Linear Regression	SVR
lofreq_indelqual/2.1.4	0.96 → 0.98	0.99 → 0.96	0.99	0.74 → 0.86
bamleftalign/1.3.1	0.76 → 0.79	0.72 → 0.78	0.59	0.77 → 0.77
vcf2tsv/1.0.0_rc3	0.09 → 0.15	-0.12 → 0.07	0.14	-0.01 → -0.04
rna_star/2.7.2b	-0.18 → 0.04	-0.22 → 0.04	0.01	-0.01 → 0.01
rna_star/2.7.5b	-0.33 → 0.09	-0.35 → 0.09	0.05	0.07 → 0.07
kraken2/2.1.1	-0.09 → 0.18	-0.02 → 0.17	0.08	-0.01 → -0.04
fasta2tab/1.1.1	0.69 → 0.80	0.72 → 0.80	0.25	0.81 → 0.78
gmx_sim/2020.4	-0.01 → 0.10	0.01 → 0.07	0.07	0.05 → 0.06

Table 5.9: R2 Scores of the models before and after HPO

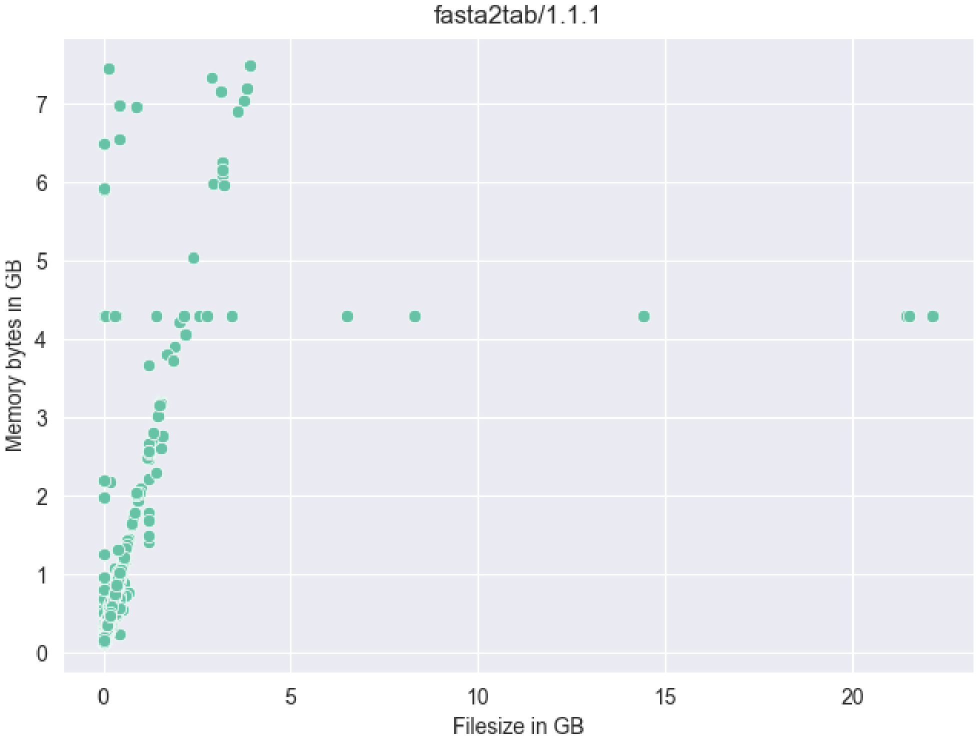


Figure 5.8: Filesize vs Memory of fasta2tab/1.1.1

5.5.1 Analysis on resource consumption

As already described in Section 4.1.2 Galaxy currently assigns each tool a fixed amount of memory. This is pre-configured in the the tool_destinations file (can be found here). Tools which are not configured in this file are assigned a default value of 1 GB for memory. Using this information and the trained models, evaluations can be done to estimate how much resources can be saved by using the trained ML models instead of the pre-assigned memory by Galaxy. For reference the tool_destinations file of the project repository was used (repository link).

Using the optimized models, the tools *kraken2/2.1.1* and *rna_star/2.7.5b* were inspected, since they have a high memory usage on average as can be taken from Table 5.8. As for both tools the Random Forest model was the best, it was chosen for analysis. The results of the analysis on the test set of both tools can be seen in Table 5.10. For *rna_star/2.7.5b* the results of the Random Forest are good. While Galaxy overallocates about 42 GB of memory on average, Random Forest assigns only about 14 GB too much (that is a decrease of about 65%). One drawback is that with the Random Forest model the percentage of failed jobs amounts to 23% compared to 12.25% from Galaxy. For *kraken2/2.1.1* Galaxy overallocates 41 GB whereas the Random Forest model overassigns only about 20 GB (a decrease of about 50%). Nevertheless the number of failed jobs increase for the Random Forest up to 44.95% compared to 25.25% from Galaxy.

		Average overallocation		**Failed jobs**	
Tool	**Assigned by Galaxy**	***Galaxy***	***RF Model***	***Galaxy***	***RF Model***
rna_star/2.7.5b	90 GB	42.52 GB	14.59 GB	12.25%	23.10%
kraken2/2.1.1	64 GB	41.38 GB	20.90 GB	25.25%	44.95%

Table 5.10: Comparison of resource allocation and failed jobs for Galaxy and Random Forest

5.6 Accuracy-Failure trade-off

As seen in Section 5.5.1, even though the Random Forest model was able to optimize the resource consumption compared to Galaxy, the percentage of failed jobs increased. To investigate the trade-off between accuracy and possible job failure the optimized Random Forest and the method described in Section 4.4 is used. Using the same tools as before and by providing a probability of 99% certainty to the tool, the results given in Table 5.11 were yielded. For both tools the average overallocation decreased. In the case of *rna_star/2.7.5b* it went up from 14.59 GB to 28.33 GB and for *kraken2/2.1.1* from 20.90 GB to 28.68 GB. This behaviour is expected since the model uses the upper bound of the prediction interval, which results in a higher estimation memory-wise. It can be observed that for both tools the percentage of failed jobs decreased significantly. For *rna_star/2.7.5b* it went down from 23.10% to 13.55% achieving a similar rate as Galaxy while having much less overallocation. At the same time for *kraken2/2.1.1* it went down from 44.95% to 35.3%. This shows the effect of the prediction with uncertainty. By setting a prediction interval with the given probability, the model is able to provide a prediction that results in a more likely job success on the first try.

	Average overallocation		**Failed jobs**	
Tool	***Galaxy***	***RF Model***	***Galaxy***	***RF Model***
rna_star/2.7.5b	42.52 GB	14.59 GB → 28.33 GB	12.25%	23.10% → 13.55%
kraken2/2.1.1	41.38 GB	20.90 GB → 28.68 GB	25.25%	44.95% → 35.3%

Table 5.11: Comparison of resource allocation and failed jobs for Galaxy and Random Forest with certainty probability 99%

6 Discussion and Conclusion

For this project the Galaxy job run dataset was analyzed and different ML methods were used to predict the memory usage of future jobs. In this context, several preprocessing methods were brought forward, including the filtering of invalid data, removal of outliers and general text processing steps to reduce the dataset size. By doing so, the Galaxy job run dataset could be reduced from 10 GB to about 2.8 GB, resulting in a decrease of approximately 70%. The findings of invalid entries in the dataset and other characteristics that were discussed in this report, raises questions about the validity of the recorded dataset. Another aspect is that it is not possible to find out if jobs in the dataset are initial runs or whether they got repeated. If it is possible, ways should be explored to track this because then this can be filtered since the goal is to predict the memory of the initial run.

The effects of filtering invalid data were examined and it was shown that all models benefit from the filtering of faulty data, resulting in significant improvements of the R2 score. Similarly, the removal of outliers was evaluated, which improved the results for Random Forest and XGB models as well.

Furthermore, it was shown that there is a relationship between the performance of the models and the correlation of the data between *Filesize* and *Memory bytes*. In general, it was apparent that a stronger correlation of the two features resulted in a more accurate and robust prediction. Equally, it became evident that for tools with low correlation, the models performed worse on average. This made one of the major drawbacks of the dataset visible: the models rely mainly on one feature, namely *Filesize*. This means, if there is no significant correlation between *Filesize* and *Memory bytes*, the models perform poorly or worse. Performance is best when there is a high correlation. In addition, the input feature *Slots*, which corresponds to the number of CPU cores allocated by Galaxy for the job, is almost the same in each dataset. Therefore, no useful information can be extracted. All in all, there are too few useful features in the dataset, so the models depend too much on a single feature. This is even more problematic since approximately 50% of the tools have a correlation less than 0.5 between *Filesize* and *Memory bytes*, meaning that prediction is difficult for many tools.

At the end, the models were optimized using Hyperparameter Optimization and a comparison to Galaxy's method of assigning fixed amounts of memory for tools was done. It was demonstrated that by using the prediction of the presented models, the overallocation of memory in some cases could be reduced by 50% to 65%. This illustrates the potential usage of such ML models in the Galaxy system to optimize resource allocation and reduce wastage. While the predictions of the models led to a higher failure rate of the jobs, a method was brought forward which uses prediction intervals to generate predictions to reduce job failure rates.

This brings us to the future work that could be done using the findings of this project. It would be of great interest to see how the proposed ML models would perform in a real scenario in the Galaxy framework. It can be investigated how it affects the memory consumption and memory waste during the execution of tools and whether this would lead to an optimization of resources. At the same time, it would be useful to look into the job failure rates of a real scenario and see how the models perform. Another research opportunity would be to examine if, instead of training one model for each tool at a time, an entire model can be generated for multiple tools. Then the performance of this model could be compared to the models brought forward in this project.

A List of abbreviations

RAM Random-Access Memory

CPU Central Proccessing Unit

SVR Support Vector Regression

XGB Extreme Gradient Boosting

RNA Ribonucleic Acid

DNA Deoxyribonucleic Acid

ML Machine Learning

VNF Virtual Network Function

NAT Network Address Translation

VM Virtual Machine

Bibliography

[1] *Coefficient of Determination, R-squared.* – https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html, accessed on 01.10.2022

[2] *Galaxy Community Hub.* – https://galaxyproject.org, accessed on 01.10.2022

[3] *Galaxy Europe.* – https://usegalaxy.eu, accessed on 01.10.2022

[4] *Galaxy Main.* – https://usegalaxy.org, accessed on 01.10.2022

[5] *Galaxy Tool Shed.* – https://toolshed.g2.bx.psu.edu, accessed on 01.10.2022

[6] *Searching for optimal parameters with successive halving.* – https://scikit-learn.org/stable/modules/grid_search.html#successive-halving-user-guide, accessed on 01.10.2022

[7] *sklearn.metrics.r2_score.* – https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score, accessed on 01.10.2022

[8] *sklearn.model_selection.HalvingGridSearchCV.* – https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.HalvingGridSearchCV.html, accessed on 27.09.2022

[9] *sklearn.preprocessing.StandardScaler.* – https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html, accessed on 01.10.2022

[10] *tool_destinations.yaml*. – https://github.com/usegalaxy-eu/infrastructure-playbook/blob/master/files/galaxy/dynamic_rules/usegalaxy/tool_destinations.yaml, accessed on 27.04.2022

[11] *Was ist ein Random Forest?*. – https://www.tibco.com/de/reference-center/what-is-a-random-forest, accessed on 29.07.2022

[12] *What are Virtual Network Functions (VNFs)?*. – https://www.thousandeyes.com/learning/glossary/vnf-virtual-network-functions, accessed on 13.09.2022

[13] *XGBoost documentation*. – https://xgboost.readthedocs.io/en/stable/#, accessed on 02.08.2022

[14] AFGAN, Enis ; BAKER, Dannon ; BATUT, Bérénice ; BEEK, Marius van d. ; BOUVIER, Dave ; ČECH, Martin ; CHILTON, John ; CLEMENTS, Dave ; CORAOR, Nate ; GRÜNING, Björn A. ; GUERLER, Aysam ; HILLMAN-JACKSON, Jennifer ; HILTEMANN, Saskia ; JALILI, Vahid ; RASCHE, Helena ; SORANZO, Nicola ; GOECKS, Jeremy ; TAYLOR, James ; NEKRUTENKO, Anton ; BLANKENBERG, Daniel: The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. In: *Nucleic Acids Res.* 46 (2018), Nr. W1, W537-W544. http://dx.doi.org/10.1093/nar/gky379. – DOI 10.1093/nar/gky379

[15] AMRO, Asma' ; AL-AKHRAS, Mousa ; HINDI, Khalil ; HABIB, Mohamed ; SHAWAR, Bayan: Instance Reduction for Avoiding Overfitting in Decision Trees. In: *Journal of Intelligent Systems* 30 (2021), 01, S. 438–459. http://dx.doi.org/10.1515/jisys-2020-0061. – DOI 10.1515/jisys–2020–0061

[16] AYDOGAN Öner: *Pearson correlation scores for all tools*. – https://github.com/Kingdogan/tool-resource-prediction/blob/main/processed_data/pearson_corr_scores.csv, accessed on 01.10.2022

[17] AYDOGAN Öner: *Tool Resource Prediction. Github Repository*. – https://github.com/Kingdogan/tool-resource-prediction

[18] Breiman, Leo: Bias, Variance , And Arcing Classifiers. In: *Technical Report 460, Statistics Department, University of California* (2000), 11

[19] Breiman, Leo: Random Forests. In: *Machine Learning* 45 (2001), Nr. 1, S. 5–32. `http://dx.doi.org/10.1023/a:1010933404324`. – DOI 10.1023/a:1010933404324

[20] Chen, Tianqi ; Guestrin, Carlos: XGBoost: A Scalable Tree Boosting System. In: *CoRR* abs/1603.02754 (2016). `http://arxiv.org/abs/1603.02754`

[21] Cheng, Fei ; Yang, Chunhua ; Zhou, Can ; Lan, Lijuan ; Zhu, Hongqiu ; Li, Yonggang: Simultaneous Determination of Metal Ions in Zinc Sulfate Solution Using UV–Vis Spectrometry and SPSE-XGBoost Method. In: *Sensors* 20 (2020), 08, S. 4936. `http://dx.doi.org/10.3390/s20174936`. – DOI 10.3390/s20174936

[22] Friedman, Jerome H.: Greedy function approximation: A gradient boosting machine. In: *The Annals of Statistics* 29 (2001), Nr. 5, 1189 – 1232. `http://dx.doi.org/10.1214/aos/1013203451`. – DOI 10.1214/aos/1013203451

[23] Hastie, Trevor ; Friedman, Jerome ; Tisbshirani, Robert: *The elements of Statistical Learning: Data Mining, Inference, and prediction*. Springer, 2017. – 587–588 S.

[24] Hyndman, R.J. ; Athanasopoulos, G.: *Forecasting: Principles and practice (2nd edition)*. – `https://otexts.com/fpp2`, accessed on 11.09.2022

[25] Kamali, Kaivan: *galaxy_job_analysis*. – `https://github.com/kxk302/galaxy_job_analysis`, accessed on 27.09.2022

[26] Kuhn, Max ; Johnson, Kjell: *Applied predictive modeling*. Springer, 2016

[27] Li, Cheng: *A Gentle Introduction to Gradient Boosting*. – `http://www.chengli.io/tutorials/gradient_boosting.pdf`, accessed on 02.08.2022

[28] Meinshausen, Nicolai: Quantile Regression Forests. In: *Journal of Machine Learning Research* 7 (2006), 06, S. 983–999

[29] NAWROCKI, Piotr ; OSYPANKA, Patryk: Cloud resource demand prediction using machine learning in the context of QoS parameters. In: *Journal of Grid Computing* 19 (2021), Nr. 2. `http://dx.doi.org/10.1007/s10723-021-09561-3`. – DOI 10.1007/s10723–021–09561–3

[30] PEDREGOSA, F. ; VAROQUAUX, G. ; GRAMFORT, A. ; MICHEL, V. ; THIRION, B. ; GRISEL, O. ; BLONDEL, M. ; PRETTENHOFER, P. ; WEISS, R. ; DUBOURG, V. ; VANDERPLAS, J. ; PASSOS, A. ; COURNAPEAU, D. ; BRUCHER, M. ; PERROT, M. ; DUCHESNAY, E.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830

[31] QUINLAN, J. R.: Induction of Decision Trees. In: *Machine Learning* 1 (1986), Nr. 1, S. 81–106. `http://dx.doi.org/10.1007/bf00116251`. – DOI 10.1007/bf00116251

[32] RICART, Narcis ; CORTADELLA, Jordi ; CASANOVA, Jonas: *Machine learning techniques for resource prediction in Nanoelectronic Circuit Design*, Universitat Politècnica de Catalunya, Diplomarbeit, 2017

[33] SAABAS, Ando: *Prediction intervals for Random Forests*. – `https://blog.datadive.net/prediction-intervals-for-random-forests/`, accessed on 11.09.2022

[34] SCHNEIDER, Stefan ; SATHEESCHANDRAN, Narayanan P. ; PEUSTER, Manuel ; KARL, Holger: Machine Learning for Dynamic Resource Allocation in Network Function Virtualization. In: *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, S. 122–130

List of Figures

3.1 Structure of a Random Forest consisting of multiple decision trees [11] 5
3.2 An example of a decision tree on the tool-resource-prediction dataset 6
3.3 Structure of the XGB model [21] 7

4.1 Excerpt of the Galaxy job run dataset 9
4.2 Plot of the true values and 90% prediction intervals [33] 16
4.3 Plot of the true values and 50% prediction intervals [33] 17

5.1 R2 Score of the models using moderate-high correlation data 22
5.2 Model performance in relation to correlation for moderate-high correlation data 23
5.3 R2 Score of the models using low-moderate correlation data 25
5.4 Model performance in relation to correlation for low-moderate correlation data 26
5.5 Filesize vs Memory bytes of bowtie2 (Galaxy main) 28
5.6 Filesize vs Memory bytes of hisat2 (Galaxy main) 29
5.7 Results for Galaxy main’s data 29
5.8 Filesize vs Memory of fasta2tab/1.1.1 32

List of Tables

4.1 The recorded job attributes 10
4.2 Relevant job attributes 11

5.1 R2 Score of the models before and after removing faulty data 18
5.2 R2 Score of the models before and after removing outliers 19
5.3 Pearson correlation for different versions of *bowtie2* 20
5.4 Tools with moderate to high correlation 21
5.5 Tools with low to moderate correlation 24
5.6 The given importance of Random Forest for the input features 26
5.7 Correlation of data from Galaxy Main [25] 27
5.8 Selected tools with their corresponding group, correlation and memory usage 30
5.9 R2 Scores of the models before and after HPO 31
5.10 Comparison of resource allocation and failed jobs for Galaxy and Random Forest 33
5.11 Comparison of resource allocation and failed jobs for Galaxy and Random Forest with certainty probability 99% 34