JACOBS
UNIVERSITY

Scalable Optimal Variable Selection

by

Ananta Raj Bhattarai

Bachelor Thesis in Computer Science

Submission: June 1, 2020 Supervisor: Prof. Peter Zaspel

Jacobs University Bremen | Department of Computer Science and Electrical Engineering

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

2020.05.15 *Ananta*

Date, Signature

Abstract

Variable selection refers to a machine learning technique where we select a subset of relevant variables or features from the data. It is widely used in machine learning to achieve high prediction accuracy and to reduce the model's training time. Many techniques are available to perform variable selection, including best subset selection, forward stepwise regression, forward stagewise regression, and least angle regression. However, the application of these mechanisms to large-scale data is infeasible because the computational complexity of training is very high.

Parallelization is a way of designing a computer program such that the sets of instructions execute in parallel. Parallelization helps to solve the scalability problem in machine learning, running array computations in parallel, and reducing the running time of the algorithms. There are several ways of achieving parallelization; however, this guided research focuses on dividing the large array into small arrays and running simultaneous calculations on these small arrays in a distributed system. Distributed parallelization allows us to efficiently remove redundant variables from large-scale data, reducing the training time of variable selection mechanisms, and minimizing computational resources.

This research aims to implement state-of-the-art variable selection techniques (least angle regression and forward stagewise regression) using Dask array, and Dask distributed for parallel scalability. It compares the performance of two approaches with respect to the quality of the selected training subsets and parallel scalability on the reference data.

Contents

1 Introduction

In today's world, machine learning models often deal with large scale datasets to train the model and to achieve good prediction performance. Such large scale data may have a very high dimension in the input. As a result of the high dimension, commonly known as the *curse of dimensionality*, there are too many variables or features that increase computational time to build the predictive model. In addition to that, the model with redundant variables may not perform well on previously unseen data. Therefore, it is essential to treat such big data and select relevant variables before carrying out the training process. Several variable selection mechanisms are available to this end, including (best subset selection, forward stepwise regression, forward-stagewise regression, and least angle regression) [7]. However, the application of these mechanisms to large-scale data is infeasible because the computational complexity of training increases with the number of variables, leading to prohibitive runtimes.

Parallelization is a way of writing or designing algorithms such that computations on the data occur parallelly in a single machine or set of machines, i.e., cluster. Parallelization is a solution to the scalability problem because it reduces the execution time of the algorithms, running tasks in parallel. There are three ways of achieving parallelization: 1) Using *multi-threading* in a single-core machine 2) Using *multi-cores* of a single machine through threads 3) Using multiple computer nodes in a network, i.e., *distributed computing*. This research focuses on achieving parallelization with distributed computing. In distributed computing, multiple machines in the cluster share the computation tasks. Dask is an open-source framework that provides high-level array and library for distributed computing in Python. Dask array divides the large data into small chunks, and Dask distributed runs simultaneous calculations on these small chunks using several worker processes spread across multiple machines.

The parallelization of variable selection algorithms helps to select the significant variables from the large-scale data efficiently. Although variable selection has been extensively studied in the field of machine learning [14] [15] [24], quantitative research on scalable variable selection remains lacking. Therefore, this research implements the two variable selection approaches (i.e., forward stagewise regression and least angle regression) [7] on the top of the Dask framework (Dask.array and Dask.distributed) [3] for parallel scalability and compares them.

Forward stagewise regression is a forward regression method that computes the regression coefficients in a stepwise increment fashion [20]. It starts with all the regression coefficients equal to zero. At each iteration, it updates the coefficient (by some small constant) of the variable highly correlated with the current residual. Forward stagewise regression allows us to select the significant variables from the data as it gives the notion about the impact of the added variables in reducing the current residual. Furthermore, it repeats simple iterations where most of the operations are matrix-vector multiplication and vector addition, array operations that Dask can parallelize.

Least angle regression is also a forward regression technique. Least angle regression finds a variable that is highly correlated with the current residual and moves its coefficient towards the least-squares value until another variable that has as much correlation with the current residual comes along the path [20]. After new variables get added to the model, it then moves in a direction equiangular between the variables. This process repeats until all the variables are included in our regression model. Least angle regression

is not greedy compared to forward stagewise regression because it moves in the direction equiangular between all added variables instead of just increasing the coefficient in the direction of one variable that is highly correlated with the current residual among added variables, hoping to find another correlated variable along the path. This allows us to produce an optimal set of significant variables. Least angle regression also involves matrix inversion operation in addition to matrix multiplication and vector addition. Although the least angle regression requires expensive operation like matrix inversion, it is computationally efficient in the sense that a new variable gets added to the model at every iteration, and it takes p iterations to reach a full solution, where p is the dimension of the input. Moreover, the array computations in the least angle regression can be parallelized with Dask.

The rest of the document is structured as follows. Section 2 describes the learning problems introduced by large data in machine learning and discusses the relevance of variable selection to solve these problems. Furthermore, it describes the motivation behind distributed parallelization and explains the distributed parallelization model implemented with Dask. Finally, it gives an insight into previous work in variable selection and states the research objectives. Section 3 gives an overview of two variable selection approaches used in this research. The implementation details and experimental setup are given in Section 4. Section 5 showcases all the results and runs an empirical analysis of them. The final section of this report is Section 6, which provides a summary of the project and discusses future work.

2 Motivation

2.1 Relevance of Machine Learning and Data

Machine Learning has been an active field of research in recent years. Many of today's state-of-the-art models in the domains of classification, image processing, robotics, speech recognition, and natural language processing are powered by machine learning algorithms [25] [12] [6]. Interest in machine learning models has evolved because of its ability to independently learn intricate patterns within the data and adapt to the new data.

Machine Learning came into rising after IBM's *Deep Blue* defeated the world champion at chess in 1977 [10]. After this tremendous achievement, research on machine learning has reached its climax. In 2006, the research on neural networks was rebranded as *deep learning* [22]. Today, deep learning is considered one of the powerful tools to improve voice recognition and image tagging. In 2011, IBM's *Watson* managed to gain victory over two jeopardy champions in a three-day showdown [9]. One year later, a neural network designed by Google learned to detect humans and cats in Youtube videos in an unsupervised manner [13]. It achieved an accuracy of 81.7%, setting the milestone in the field of classification. In 2015, Google's *AlphaGo* defeated the professional player at Go, which is considered the most difficult board game in the world [4].

Data is the key to the success of machine learning algorithms. The quality, as well as the quantity of the data, determines the performance of the predictor. While data plays a vital role in machine learning, it has also been recognized that large scale data brings inefficiencies in the learning process of machine learning algorithms [11], which is discussed in Section 2.2.

2.2 Learning Problem in Machine Learning and Variable Selection

Machine learning models generally, but not all, are high-order polynomial functions of the number of variables or even exponential functions, which leads the training time to increase exponentially with the number of variables. In addition to that, large-scale data has a very high dimension in the input, which may bring overfitting issues. Overfitting is defined as a very low training error but high testing error.

We now introduce the concepts of least-squares regression [7] to elaborate on learning problems in machine learning and to discuss how variable selection algorithms fit into the problem described in this section.

Least-squares regression is a commonly used linear modeling method in the field of machine learning. Least-squares regression approximates the multidimensional data set using the formula given by

$$\hat{y} = \beta_0 + \sum_{j=1}^{D} x_j \beta_j, \tag{1}$$

where $\hat{y}$ is the response variable, x_j's are the independent or predictor variables, β_j's are the regression coefficients and D is the total number of independent variables in the model. Least-squares regression uses all the predictor variables from the training set and finds the coefficients that best fits the data. In other words, it finds the coefficients that

minimize the residual sum of squares (RSS) given by

$$\text{RSS} = \sum_{i=1}^{N} (y_i - \hat{y}_i), \tag{2}$$

where y's are the true values, $\hat{y}$'s are the predicted values by the linear model, and N is the total sample size. However, the least-squares model may not be the best if we move to a higher dimension and have a large number of independent variables than the sample size. Moreover, the linear approximation may not be enough to describe the relationship between independent variables and output variables. In such cases, non-linear mapping of independent variables into high-dimensional feature space is often done so that the output variables can be described as linear combinations of non-linear terms. The non-linear transformation grows the dimension of the input from $\mathbf{X} \in \mathbb{R}^{N \times d}$ to $\mathbf{X} \in \mathbb{R}^{N \times D}$, where $\mathbf{X}$ is the input matrix, N is the total sample size, d is the dimension of input before non-linear transformation and D is the dimension of the input after non-linear transformation. High dimension mapping increases the training complexity because we now have to solve for D regression coefficients. In addition to that, it increases the risk of overfitting because the linear model may fit the training data perfectly. Overfitting degrades the performance of the machine learning model, capturing random noise in the training data.

Variable selection techniques are available to set some of the coefficients to zero and to select only predictor variables that exhibit the most substantial effects. Variable selection decreases the computational complexity of training and maximizes the performance of the machine learning predictor. The scope of variable selection is not just limited to least squares regression. Moreover, the generalization performance of powerful deep learning models like neural networks increases, and the training complexity decreases after applying the variable selection mechanism.

2.3 Making large-scale ML feasible by Distributed Parallelization

In the previous section, we discussed how the variable selection algorithms help to combat overfitting issues and to reduce the computational complexity of training algorithms. There are two types of variable selection techniques: Filter and Wrapper [19]. Filter techniques look at the variables' inherent properties in order to determine the significant variables, whereas wrapper techniques train the model to perform variable selection. The complexity of wrapper techniques increases with the number of variables in the data. In our work, we use forward stagewise regression and least angle regression, which fall under wrapper techniques. Therefore, large-scale data, along with a high computational complexity of these techniques, leads to prohibitive runtimes to carry out the selection process. Hence, the success of machine learning lies in the parallelization of algorithms to reduce the training time, minimizing computational resources.

The rest of the section gives an overview of different types of parallelization and states the motivation behind distributed parallelization.

Multi-threading parallelization in a single core machine is achieved by executing multiple threads at the same time, each thread carrying one instruction. Multiple threads within the same process can communicate with each other and share the same memory. However, only one thread can have access to memory at once. Hence, the scheduler decides the execution of the thread. In a single-core CPU, the state of the active thread must be saved

before switching to the next thread. It is commonly known as *context switching*. The idea of using multi-threading in a single core machine is not feasible for complex algorithms because context switching is an expensive procedure since the scheduler spends an extra amount of time to suspend the active thread, to save active thread's state and to bring next thread into execution. In addition to that, most often, it also does not lead to performance improvements.

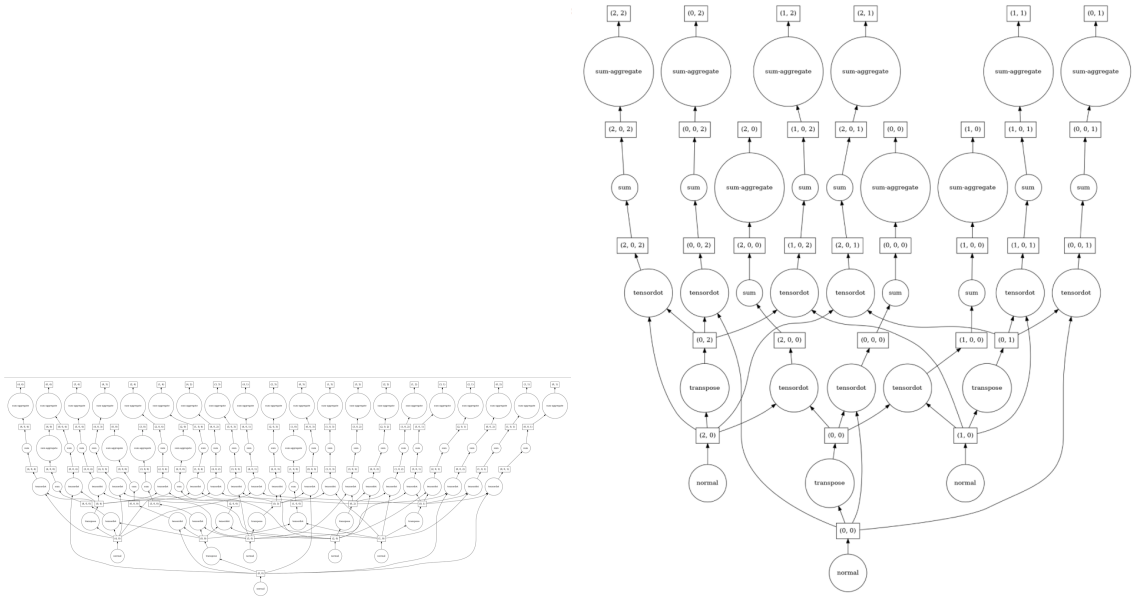
Multi-core parallelization in a single machine is carried out by executing instructions in parallel where each core handles separate execution of instructions. Therefore, each thread runs in parallel in each core. In this setting, multiple cores have access to the same memory, and the communication between the threads is done through the memory bus. While the data exchange through memory bus is fast, the notion of using multi-cores in a single machine requires a large number of CPU cores and a considerable amount of memory for large scale data, which increases the computational cost in terms of resources.

Distributed parallelization uses the CPU core of multiple machines in the cluster to operate simultaneously on the data. The machines communicate via the network, and they have their own memory. Distributed parallelization solves the problem of requiring a huge amount of memory in a single machine by distributing large scale data across multiple machines in the cluster. Therefore, it allows us to efficiently remove redundant variables from large-scale data, reducing the training time of variable selection mechanisms, and minimizing computational resources.

2.4 Distributed Parallelization with Dask

Dask is a library for parallel computing in Python [3]. It provides user interfaces, including array (Dask.array), for parallel computations. Dask array cuts up the large array into small NumPy arrays using blocked algorithms, where small NumPy arrays are called chunks [3]. The operations on a single Dask array trigger many computations on each of the small NumPy arrays. Dask array functions encode the algorithms in the form of a task graph. The task graph consists of nodes with edges between nodes if one task depends on data produced by another [3]. The number of chunks influences the size of the graph. A large number of chunks result in a massive number of nodes on the task graph. The relationship between the number of chunks and the size of the task graph for simple matrix multiplication is showcased in Figure 1.

Dask also provides a distributed computing library (Dask.distributed), which is a dynamic task scheduler [3]. The distributed scheduler manages the requests of the client and coordinates the actions of several Dask worker processes spread across multiple machines [3]. The computation with Dask distributed proceeds in the following stages: 1) The user generates the task graph using Dask array functions on the client and submits these tasks to the distributed scheduler. 2) The distributed scheduler inserts all these tasks into its data structures and assigns them to various workers in a way that leverages parallelism. 3) The workers communicate with each other via the network to collect data dependencies and run required functions on the data. 4) The workers store the result in their memory after the computations are finished and report the task completion status back to the distributed scheduler. 5) The distributed scheduler reports the task completion status back to the user, and the user fetches the result from the workers through the distributed scheduler. The flowchart of computation with Dask distributed is shown in Figure 2.



(a) Number of chunks $= 200$ (b) Number of chunks $= 400$

Figure 1: Relationship between size of task graph and number of chunks for simple matrix multiplication

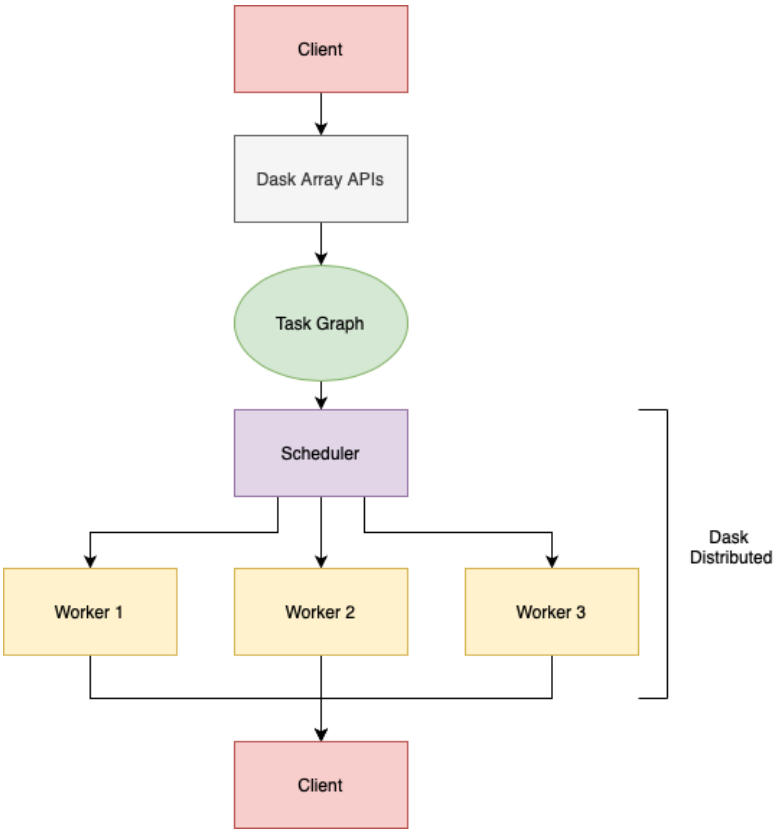


Figure 2: Flowchart of computation with Dask distributed

2.5 Related Work in Scalable Variable Selection

Parallel Variable selection has been actively explored in recent years [16] [21] [23]. Our key goal of this research to efficiently find a subset of variables from large-scale data using distributed parallelization is inspired by the work of Jonathan Wang in 2017 [23]. They used Sequential Backward Selection to select the optimal set of variables and parallelized sequential selection across 50 workers to utilize a high-performance computing platform. The results shown in the paper are inspiring. They were able to reduce the training time from over 18 hours (65020 seconds) to less than an hour (2727 seconds) while achieving the same set of variables. In our work, we parallelize the other two state-of-the-art approaches to variable selection on the top of the Dask framework. This is a new research idea that has not been previously explored to the best of our knowledge.

2.6 Research Objectives

Our research focuses on making state-of-the-art variable selection algorithms scalable, achieving distributed parallelization with the use of Dask array and Dask distributed. This report's principal goal is to give an overview of two variable selection techniques and to compare both approaches in terms of parallel scalability and the number of variables selected.

Considering the goals, the objectives of this research are to implement least angle regression and forward stagewise regression using Dask array and Dask distributed, to optimize the parallel performance of both algorithms by appropriate scheduling of the involved tasks and to compare the results of the algorithms with respect to the quality of the selected training subsets and with respect to parallel scalability on reference data.

3 Variable Selection Methods

3.1 Mathematical Setting

In regression setting, the matrix of predictor variables is denoted by $X \in \mathbb{R}^{n \times p}$, where the columns $x_1, x_2, ...x_p \in \mathbb{R}^n$ are the set of independent variables. The regression coefficients are represented as $\beta_1, \beta_2, ...\beta_p$. Similarly, the true outcome variables and predicted outcome variables by the model is denoted by $y \in \mathbb{R}^n$ and $\hat{y} \in \mathbb{R}^n$ respectively. In theory, the algorithms presented in this research assume that the independent variables have zero mean and unit length, and the outcome variable has zero mean.

$$\sum_{i=1}^{N} y_i = 0 \quad \sum_{i=1}^{N} x_{ij} = 0 \quad \sum_{i=1}^{N} x_{ij}^2 = 1 \quad \text{for j} = 1, 2, ..., p \tag{3}$$

We ignore the bias or intercept in our algorithms because it is equal to zero when both the independent and output variables are standardized to have mean zero. From now on, we use a bold alphabet to represent a vector and bold uppercase alphabet to represent a matrix. In addition to that, we follow the notations described in this section, unless explicitly stated otherwise.

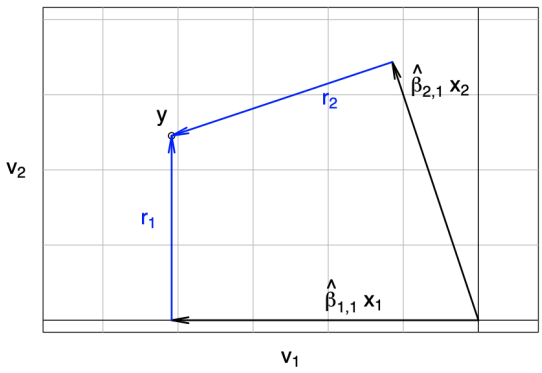
3.2 Forward Stagewise Regression

Forward regression techniques build an estimator in a stepwise fashion by finding the variable most correlated with the residual at each iteration. Forward Stagewise regression is also one of the forward regression techniques. Forward stagewise produces the sequence of regression coefficients by iteratively increasing the coefficient of the variable that has the most substantial impact on reducing the residual error. A detailed description of the algorithm is given in Algorithm 1.

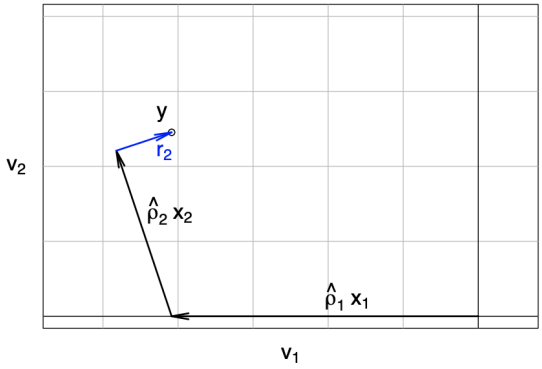
Algorithm 1 Forward stagewise regression

Set all the coefficients to zero $(\beta_1, \beta_2, ...\beta_n) = 0$ and $\hat{\mathbf{y}} = \mathbf{0}$
Set **residual (r)** $= \mathbf{y}$
Set active variable list $=$ {empty}
for $j \leftarrow 1$ to $iterations$ **do**
  correlation (c) $= \mathbf{X}^T\mathbf{r}$
  $i =$ index of maximum element in $|\mathbf{c}|$
  Put i in active variable list
  if $\mathbf{c}[i] < 0$ **then**
    direction of correlated variable $(sign) = -1$
  else
    direction of correlated variable $(sign) = 1$
  end if
  $\hat{\mathbf{y}} = \hat{\mathbf{y}} + \epsilon \cdot sign \cdot \mathbf{x}_i$
  $\beta_i = \beta_i + \epsilon \cdot sign$
  $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$
end for

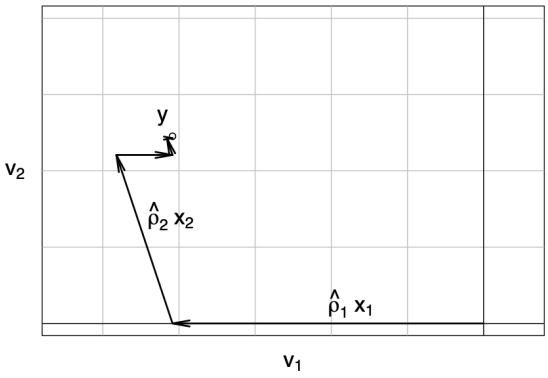
The ϵ in Algorithm 1 is a learning rate to increase the magnitude of the coefficients at each iteration and differs with respect to training data. Forward stagewise greedily selects the variable i that is highly correlated with the current residual at every iteration. However, the greediness in selecting the variable is balanced by the small value of ϵ. The critical difference between the large value of ϵ and a small value of ϵ is slow learning. With the large value of ϵ, the algorithm takes a big step in the direction of the current correlated variable while updating the fitted model. This may eliminate the possibility of other variables being highly correlated with the residual. Therefore, the algorithm may get stuck in the local minimum.



(a) Forward Stagewise $j = 1$



(b) Forward Stagewise $j = 2$



(c) Forward Stagewise after several iterations

Figure 3: Geometrical Interpretation of Forward Stagewise Regression (Figure taken from [5]

The geometrical interpretation of forward stagewise regression with $\epsilon = \mathbf{x}_i^T(\mathbf{y} - \hat{\mathbf{y}})$ is shown in Figure 3. The independent variables ($\mathbf{x}_1$, $\mathbf{x}_2$) and the outcome value $\mathbf{y}$ are projected into the plane of orthonormal unit vectors ($\mathbf{v}_1, \mathbf{v}_2$). In the first iteration, the algorithm picks $\mathbf{x}_1$ over $\mathbf{x}_2$ because $\mathbf{r}_1$ is shorter than $\mathbf{r}_2$ and increases the coefficient of $\mathbf{x}_1$ by $\epsilon = \mathbf{x}_i^T(\mathbf{y} - \hat{\mathbf{y}})$. In the second iteration, $\mathbf{x}_2$ is picked as it is the highly correlated variable with the current residual. Similarly, the coefficient of $\mathbf{x}_2$ is increased by $\epsilon = \mathbf{x}_i^T(\mathbf{y} - \hat{\mathbf{y}})$. This procedure continues for several iterations and finally we arrive at the full least-squares solution.

The forward stagewise requires many steps in order to reach the full least-squares solution. However, most of the computation involves matrix-vector multiplication and vector

addition that can be parallelized using Dask array and Dask distributed. The parallelization allows us to reduce the training time in order to reach the full least-squares solution.

3.3 Least Angle Regression

Least angle regression is a forward model selection technique. It is closely related to the lasso and forward stagewise regression. The lasso jointly minimizes the least-squares error and the L_1 norm of the coefficient vector [7]. Mathematically, the loss function minimized by lasso is given by

$$L = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \sum_{i=1}^{N}|\beta_i| \leq t, \tag{4}$$

where t controls the learning process and shrinks some of the coefficient values to zero. Some quadratic solvers can be used to solve the above minimization problem; however, Bradley Efron and the team developed the least angle regression in 2004 and showed that it could be used to solve the minimization problem with a minor modification to the original least angle regression algorithm [20]. The least angle regression algorithm is computationally efficient than quadratic solvers because it takes advantage of the parameter solutions, i.e., γ (see Equation 5) being linear that can be computed in steps. In this research, we stick to the original algorithm since our main goal is to efficiently remove redundant variables that do not have a larger contribution to the model performance rather than focusing on the regularization of the model.

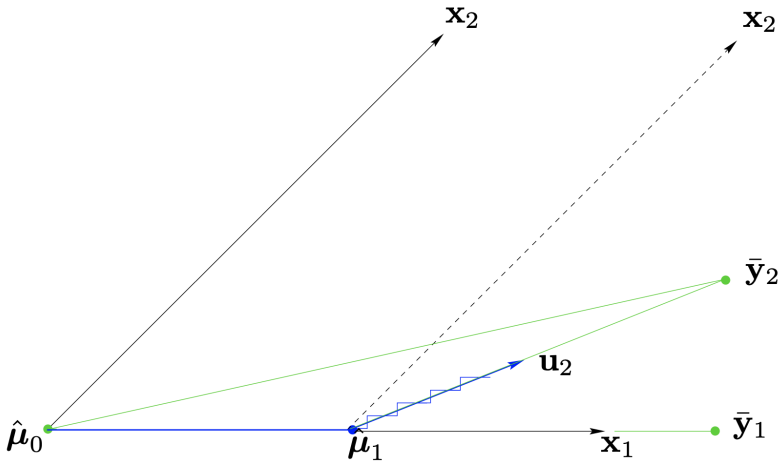


Figure 4: Geometrical Interpretation of Least Angle Regression (Figure taken from [20])

In least angle regression, the selection of the variable is done in a similar manner as with forward stagewise regresion. The variable that is highly correlated with the residual at each step is added to the model. However, when the new variable gets added to our model, we move in a direction equiangular between all the variables that are in our active set. Figure 4 demonstrates least angle regression in the case of two independent variables $(\mathbf{x}_1, \mathbf{x}_2)$. $\bar{\mathbf{y}}_2$ is the projection of target value $\mathbf{y}$ into $(\mathbf{x}_1, \mathbf{x}_2)$-plane. The prediction model $\hat{\mathbf{y}}$ is denoted by $\hat{\mathbf{u}}$ in Figure 4. It starts with $\hat{\mathbf{u}}_0 = \mathbf{0}$. In the first iteration, $\mathbf{x}_1$ has greater correlation with the residual $(\bar{\mathbf{y}}_2 - \hat{\mathbf{u}}_0)$. Therefore, $\mathbf{x}_1$ is included in the active set and the next estimate is $\hat{\mathbf{u}}_1 = \hat{\mathbf{u}}_0 + \gamma_1 \cdot \mathbf{x}_1$, where γ_1 is computed in such a way that residual $(\bar{\mathbf{y}}_2 - \hat{\mathbf{u}}_1)$ bisects the angle between $\mathbf{x}_1$ and $\mathbf{x}_2$. In the second iteration, $\mathbf{x}_1$ and

$\mathbf{x}_2$ both have equal correlation with the current residual ($\overline{\mathbf{y}}_2 - \hat{\mathbf{u}}_1$). Hence, $\mathbf{x}_2$ is picked because it is a unique variable not in the active set. Then, the estimate update is given by: $\hat{\mathbf{u}}_2 = \hat{\mathbf{u}}_1 + \gamma_2 \cdot \hat{\mathbf{u}}_2$, where $\hat{\mathbf{u}}_2$ is the unit bisector ($\hat{\mathbf{u}}_2 = \overline{\mathbf{y}}_2$) in the case of two independent variables. The generalized version of algorithm for p variables is given in Algorithm 2.

Algorithm 2 Least angle regression

Start with $\hat{\mathbf{y}} = \mathbf{0}$
Set **residual (r)** $= \mathbf{y}$
Set active variable list $=$ {empty}
for $j \leftarrow 1$ to p **do**
  correlation (c) $= \mathbf{X}^T\mathbf{r}$
  $i =$ index of maximum element in $|\mathbf{c}|$ that is not in the active variable list
  Put i in active variable list
  if $\mathbf{c}[i] < 0$ **then**
    direction of correlated variable $(s_i) = -1$
  else
    direction of correlated variable $(s_i) = 1$
  end if
  $\mathbf{X}_A = (s_i \cdot \mathbf{x}_i)_{i\in}$(active variable list)
  $\mathbf{G}_A = \mathbf{X}_A^T\mathbf{X}_A$
  $A_A = (\mathbf{1}_A^T\mathbf{G}_A^{-1}\mathbf{1}_A)^{-1/2}$, where dimension of $\mathbf{1}_A$ = length of active variable list
  $\mathbf{w}_A = A_A \cdot \mathbf{G}_A^{-1}\mathbf{1}_A$
  $\mathbf{y}_A = \mathbf{X}_A\mathbf{w}_A$
  $\hat{\mathbf{y}} = \mathbf{y}_A + \gamma \cdot \mathbf{y}_A$
  $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$
end for

The γ in Algorithm 2 is the smallest possible value that is calculated such that the next variable gets added to our model in the next iteration. Mathematically, γ is given by

$$\gamma = \min^{+}_{k\in A^C} \left\{ \frac{C - c_k}{A_A - a_k}, \frac{C + c_k}{A_A + a_k} \right\}, \tag{5}$$

where C is the maximum absolute value in $\mathbf{c}$, c_k is the value at kth row of $\mathbf{c}$, a_k is value at kth row of $\mathbf{X}^T\mathbf{y}_A$ and A_A is given in Algorithm 2. The "min+" in the above equation indicates that out of all the choices of k, only the minimum over positive elements is taken. The mathematical proof of the algorithm is in the original paper [20].

Least angle regression seems computationally expensive compared to forward stagewise regression since it involves matrix inversion step and calculations to compute γ at each iteration. However, considering the linear behavior of the solution path between each step and full least-squares solution can be produced in p steps, it is computationally efficient. Least angle regression is not greedy in a sense that it moves in a direction equiangular between all the variables rather than moving along the direction of just one variable, hoping to find another highly correlated variable along the path. Moreover, the computations on the array can be divided into small tasks using Dask array, and the small tasks can be distributed among many workers to run in parallel using Dask distributed. This allows us to select an optimal set of significant variables in a very efficient way.

4 Experimental Details

4.1 Implementation Details

We, as a user running Jupyter Notebook, create a cluster, calling the *LocalCluster()* method from Dask distributed. The *LocalCluster()* method gives us the flexibility of specifying number of workers as an argument. The *LocalCluster()* object is then passed as an argument to *Client()* method, which sets up a distributed scheduler in the local process and several processes running single threaded workers. Therefore, the task graphs can be submitted to the cluster, invoking *Client.persist()* and *Client.compute()* methods on the client.

In the second phase, we load the data using NumPy. The data is sent to the cluster using *Client.scatter()* method. This method returns a *future* object pointing to the data. However, the data may be on just one worker, rather than spread out. Therefore, we retrieve the future object into the Dask array using *Dask.array.from_delayed()* method and cut it into small chunks, passing *chunks* as an argument. After this, we spread out the chunks into the workers, calling *Client.rebalance()* method. Hence, the operations on a single Dask array triggers many computations on each of the small arrays in parallel.

The algorithms are implemented using Python objects and Dask array. Dask array is used where the array operations are required. We submit the operations on Dask array to the cluster using *Client.persist()*, where the operations are passed as an argument. However, some of the array operations like index assignment are not supported by Dask array. Therefore, we use the NumPy array in such cases and convert the NumPy array to Dask array before submitting the tasks to the cluster. The operations on Python objects, such as inserting items on the list, are not submitted to the cluster. If such operations depend on the collective results of workers, the user fetches the results from the workers and carries out the operations on the client-side. Nevertheless, there is no guarantee that all workers finish the tasks at the same time. So, the workers that compute the tasks faster remain idle until remaining workers finish the tasks. To further clarify, the operations that are not supported by Dask array are computed on the client-side and are sequential. The detailed description of the APIs mentioned in this section can be found on the Dask website [3].

4.2 Dataset Description and Pre-Processing

We pick the MD17 data set [1] [17] [2], which has molecular dynamics trajectories of small molecules, for our experiments. It has several molecules of slightly different geometries (i.e., positions of the atoms), which lead to different energy values. In our work, we only use geometries of "Malonaldehyde," molecules represented in SLATM representation [8]. Therefore, we have about one million training samples, each with 3121 dimensions in the input and one dimension in the output.

We pre-process the data before feeding into our variable selection algorithms. The independent variables are scaled to have mean zero and unit variance. In addition to that, the output vector is also scaled to have mean zero. Mathematically,

$$\sum_{i=1}^{N} y_i = 0 \qquad \sum_{i=1}^{N} x_{ij} = 0 \qquad \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \bar{x}_j) = 1 \qquad \text{for j} = 1, 2, ..., p \tag{6}$$

where y_i's are the output values, x_{ij} are the values in the input matrix , $\bar{x}_j$'s are the mean of independent vector, N is total sample size and p is dimension of the input. The *scikit-learn* standard scalar library is used to scale the data.

4.3 Evaluation Metric

4.3.1 Variable Selection

In the machine learning training process, the data is split into training and validation sets. The model is trained using the training set and evaluated on the validation set. The same process applies to variable selection algorithms as well. After a new variable is added to our model, the model's generalization error is evaluated to determine the significance of the variable in the data. The generalization error is given by the equation

$$\text{Mean Squared Error} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{7}$$

where y_i's are the true output values in the validation set, $\hat{y}_i$'s are the model's predicted values in the same validation set, and N is the sample size. However, in our research, we record the generalization error after the entry of a new variable in the model until we reach the desired number of variables since our main focus is to efficiently remove redundant variables from the data instead of increasing prediction performance.

We use 70% of the data for training and rest for validation. At first, the generalization performance is evaluated until we reach 100 variables in the model. In the second phase, we analyze the generalization error trend until we have 300 variables. Finally, we do the same until we reach 500 variables. This ensures fair comparison between both techniques in terms of time complexity as forward stagewise regression may take millions of steps to include 1000 variables, whereas the least angle regression requires 1000 iterations for the same dataset [20]. Furthermore, we analyze stagewise regression for different learning rate values, i.e., epsilon (ϵ).

4.3.2 Parallel Scalability

Dask array functions eventually call NumPy functions to perform operations on chunks. However, NumPy linear algebra routines are often multi-threaded. This causes problems because Dask runs as many parallel tasks as the number of logical cores in the cluster. Therefore, it is necessary to avoid oversubscribing threads. This is achieved by setting environment variables *OMP_NUM_THREADS*, *MKL_NUM_THREADS*, and *OPENBLAS_NUM_THREADS* to 1 before the start of the program.

Chunk size plays an essential role in the Dask because fewer chunks lead to less parallelization, whereas more chunks lead to scheduler overhead. Chunk size should be selected in such a way that a chunk should be small enough to fit comfortably in memory, and Dask can have many chunks in memory at once [3]. However, a chunk must be large enough that computations on that chunk take significantly longer than the 1ms overhead per task that Dask scheduling incurs [3].

To analyze the parallel scalability of our implemented algorithms, we calculate the speedup performance given by

$$\text{Speedup} = \frac{\text{Running time for 1 worker}}{\text{Running time for } n \text{ workers}} \tag{8}$$

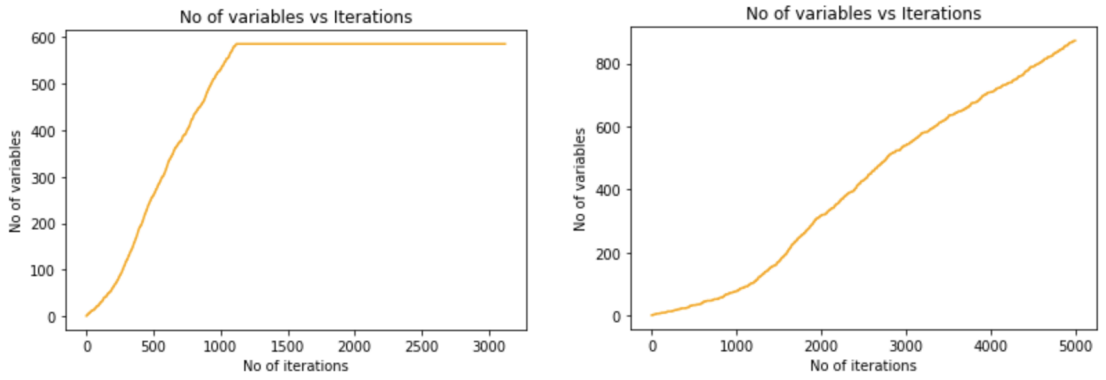
where n is varied from 1 to 30. The more workers, in principle, the more CPU cores are used to carry out the calculation. The running time for the first 50 iterations of the algorithms is recorded, changing the number of workers from 1 to 30 and keeping chunk size constant. We run for only 50 iterations because this gives us a general idea about the parallel scalability of the algorithms. Although it is tough to determine the perfect chunk size, we evaluate the speedup for different chunk sizes to analyze the effects of chunk size on the parallelization of the algorithms.

All the experiments were carried on a Computer node Titlis with *sysGen/SUPERMICRO SuperServer SYS-1029GQ-TRT*, *2x Intel® Xeon® Scalable Processor "Cascade Lake" Silver 4210 2.20 GHz 10-Core*, *192GB (12x 16GB) DDR4 PC2933*, *960GB Intel SSD D3-S4510 960GB 2.5" SATA* and *NVIDIA® Quadro RTX 4000 GPU*.

5 Results and Analysis

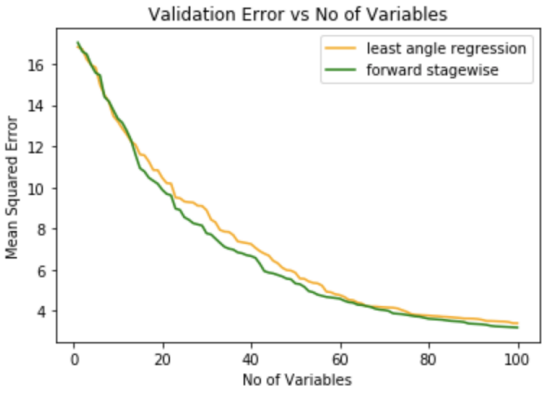
5.1 Variables Selected

The comparison of the generalization error trend against the number of variables between forward stagewise regression and least angle regression is shown in Figure 6. For the first 100 variables, generalization error decreases monotonically. However, after this, there is a slow decline in the generalization error. This suggests that the first 100 variables in the model have a very high significance in the data. The fast decrease in the error of the forward stagewise regression in Figure 6(a) compared to Figure 6(b) is due to the larger learning rate i.e ϵ. This is also verified by Figure 5(a) and Figure 5(b) since it took around 400 iterations to include 100 variables when $\epsilon = 0.05$ whereas it took around 1000 iterations to include 100 variables when $\epsilon = 0.01$. The error trend of both forward stagewise and least-angle regression is similar after the new variable is added to the model. This gives us the intuition that both techniques are including the same set of variables along the path. However, forward stagewise requires more iterations than least angle regression to produce the same set of variables. Least angle regression takes 100 iterations to produce 100 variables, whereas forward stagewise took more, which can be visualized in Figure 5(a). Figure 5(a) also shows that there is no variable included after 1100 iterations. The reason is due to the larger learning rate; the model converged too quickly to a sub-optimal solution. This means forward stagewise regression took a larger step in the direction of the correlated variable that eliminated the possibility of other variables being highly correlated with the residual. However, if the learning rate is too low, it takes a longer time to train the model. In Figure 5(b), even after 4000 iterations, more than the dimension of the input, forward stagewise with $\epsilon = 0.01$ included around 600 variables in the model. Therefore, finding a good learning rate is very crucial in the case of forward stagewise regression. As there are not any mathematical formulas to calculate the best learning rate, the only way is to evaluate the generalization error with different learning rates. Our results of both variable selection techniques (with different learning rates in the case of forward stagewise) show that the 100 variables are crucial in the MD17 dataset.

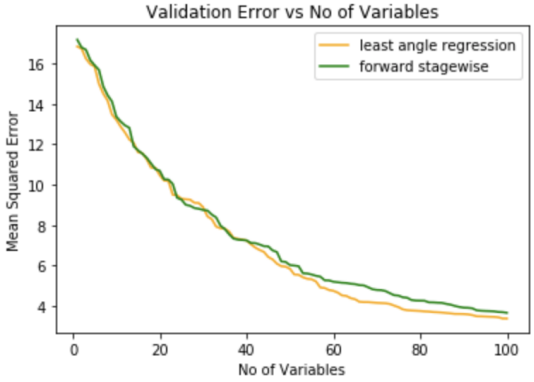


(a) $\epsilon = 0.05$ (b) $\epsilon = 0.01$

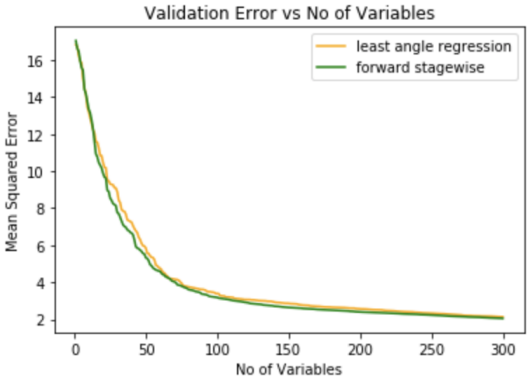
Figure 5: Comparison of Number of variables vs Number of iterations for Forward Stagewise Regression with different learning rates



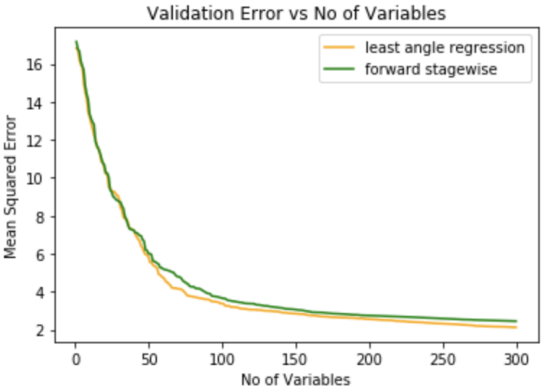
(a) No of variables $= 100$, $\epsilon = 0.05$



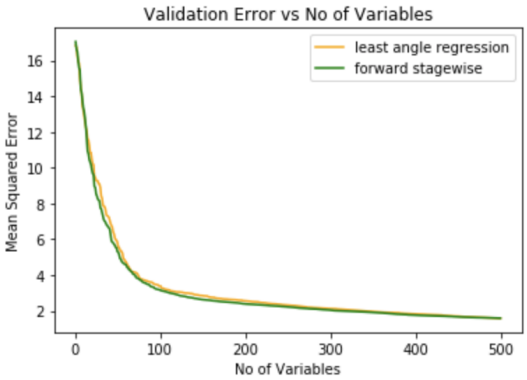
(b) No of variables $= 100$, $\epsilon = 0.01$



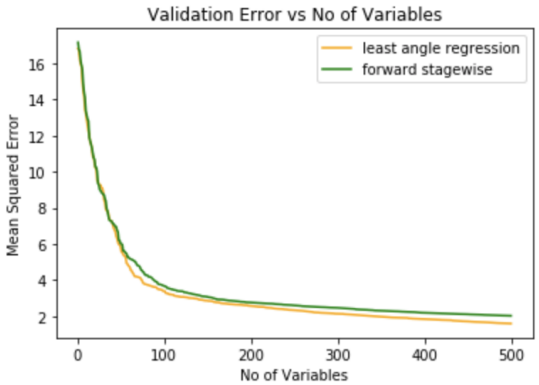
(c) No of variables $= 300$, $\epsilon = 0.05$



(d) No of variables $= 300$, $\epsilon = 0.01$



(e) No of variables $= 500$, $\epsilon = 0.05$



(f) No of variables $= 500$, $\epsilon = 0.01$

Figure 6: Comparison of Generalization error vs Number of variables between Forward Stagewise (with different learning rates) and Least Angle regression

5.2 Parallel Scalability

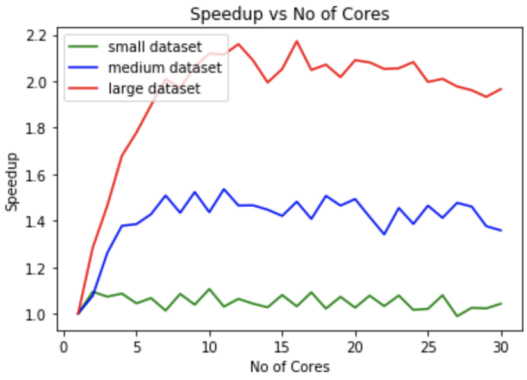
In this section, we refer to three different datasets. They are not three different datasets, but subsets of our dataset that differ in size. The information on three datasets is given in Table 1. In addition to that, we analyze the results with respect to three different chunk sizes. The information on chunk sizes is tabulated in Table 2. Figure 7 shows the comparison of speedup performance against the number of workers (cores) between forward

Table 1: Subsets Description of Original Data

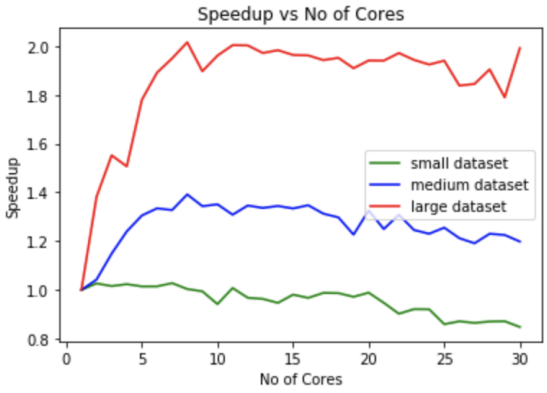
Name	Total samples	Number of Variables
Small Dataset	100000	3121
Medium Dataset	500000	3121
Large Dataset	993237	3121

Table 2: Information on 3 different chunk sizes

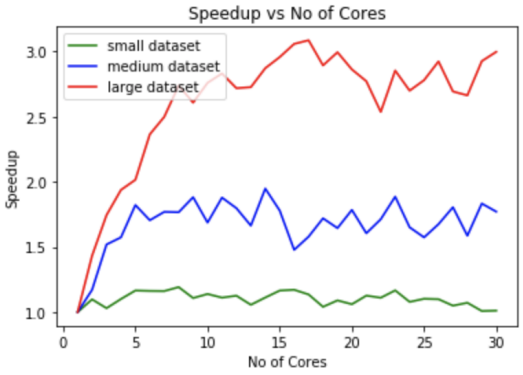
Name	Chunk Size			
	Train **X**	Train **y**	Validation **X**	Validation **y**
Small Chunk Size	(7000, 3121)	(7000, 1)	(7000, 3121)	(7000, 1)
Medium Chunk Size	(120000, 3121)	(695265, 1)	(100000, 3121)	(297972, 1)
Large Chunk Size	(200000, 3121)	(695265, 1)	(100000, 3121)	(297972, 1)



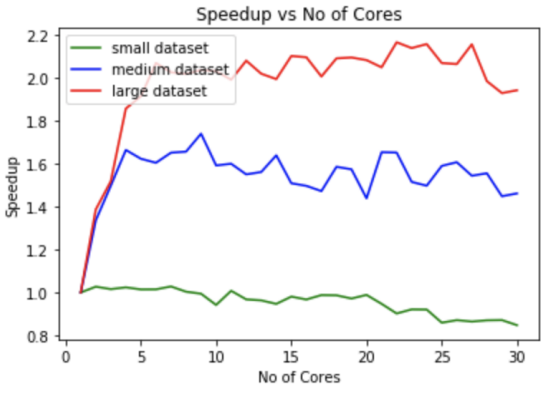
(a) Forward stagewise with data loading time



(b) Least angle with data loading time



(c) Forward stagewise without data loading time

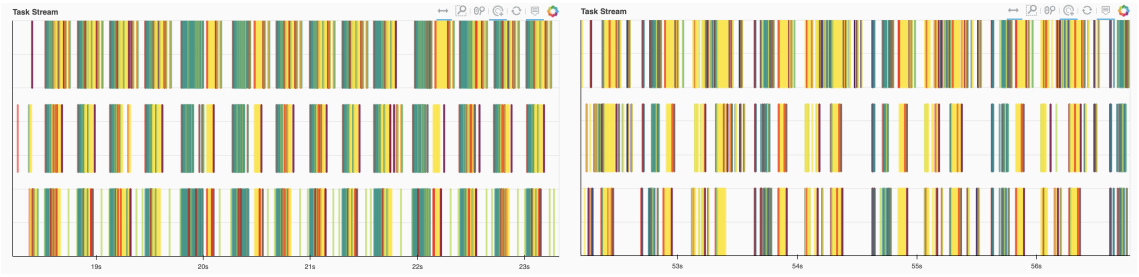


(d) Least angle without data loading time

Figure 7: Comparison of Speedup vs. Number of workers (cores) between Forward Stagewise Regression and Least Angle regression on different datasets

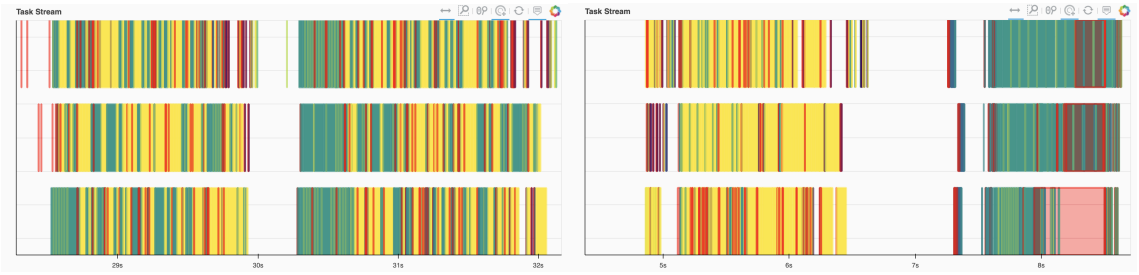
stagewise regression and least angle regression on three different datasets. Chunk size was kept constant for all datasets. Therefore, the small dataset has fewer tasks than a large dataset. We used a small chunk size to have some level of parallelism for the small

dataset as well. As we can see, the speedup for large datasets increases monotonically with an increasing number of cores compared to the other two datasets. This shows that the effect of distributed parallelization can be seen as the dataset gets larger because more tasks in the large dataset can be divided among more workers to run in parallel. However, there is a point of stagnation in performance improvement for all datasets. If we deploy more workers than required, the fixed number of tasks gets divided among a large number of workers, which causes too much communication between the workers. In addition to that, some workers take very less amount of time to finish the tasks that lead to scheduler overhead. The speedup trend for both algorithms is almost similar. However, forward stagewise has a significant increase in speedup while excluding the loading time of the data into the cluster. The least angle regression is difficult to parallelize compared to forward stagewise in the case of small chunk size because it has complex operations like matrix inversion, which need more communication between the workers than matrix multiplication and vector addition. Figure 8 shows the visualization of the Dask diagnostic dashboard captured randomly during the execution of the code.



(a) Forward Stagewise, Small Dataset, No of workers = 3

(b) Least Angle, Small Dataset, No of workers = 3



(c) Forward Stagewise, Big Dataset, No of workers = 3

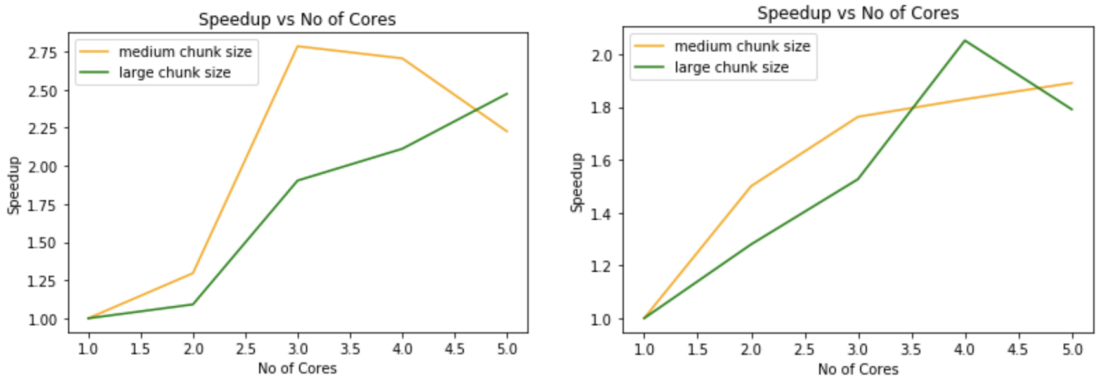
(d) Least Angle, Big Dataset, No of workers = 3

Figure 8: Figures show the activity of workers in the cluster over time, where one line corresponds to one worker and every individual rectangle corresponds to one task. The red rectangles are communication between the workers and white spaces are the idle time of the workers.

Figure 8(a) and 8(c) show the forward stagewise regression on small and big datasets, respectively, with three workers and a small chunk size. Figure 8(b) and 8(d) show the least angle regression on small and big datasets, respectively, with three workers and a small chunk size. We have two tasks in total yellow and green if we consider the chunk to be 1. Fewer chunks mean less number of yellow and green tasks. Hence, the small dataset has very few tasks compared to the large dataset. The increasing number of workers does not affect the speedup performance in the small dataset since there is more communication between the workers and scheduler overhead with just three workers.

However, if we keep the number of workers constant and increase chunks, there are more tasks in the share of workers (Figure 8(c)). Here, the scheduler overhead is somewhat reduced. However, there is not a significant increase in the speedup performance as scheduler overhead still exists (white spaces), and there is too much communication between workers (red rectangles). Least angle regression is difficult to parallelize due to high scheduling overhead and more communication between workers compared to forward stagewise for the same chunk size (Figures 8(c) and 8(d)).

Although Figure 7 shows that our implementation of the algorithms is scalable, there is not a significant increase in the speedup performance with an increased number of workers. Achieving the speedup of 3 using 30 workers is considered as an inefficient use of parallel hardware. The reason behind this is due to the small chunk size. A large number of chunks increases the size of the task graph. When there are millions of tasks, the overhead of constructing the task graph can be considerable. In addition to that, Dask eventually calls NumPy functions on small chunks to perform computations, and NumPy is very fast on small arrays. Therefore, when the computations are too fast, there is a significant scheduler overhead. The solution is to increase the chunk size so that the tasks graph gets small, and computations on the chunks take a significantly longer time to finish. Figure 9 shows some improvement over the speedup performance with medium and large chunk size on the large dataset. The loading time of the data into the cluster was excluded.



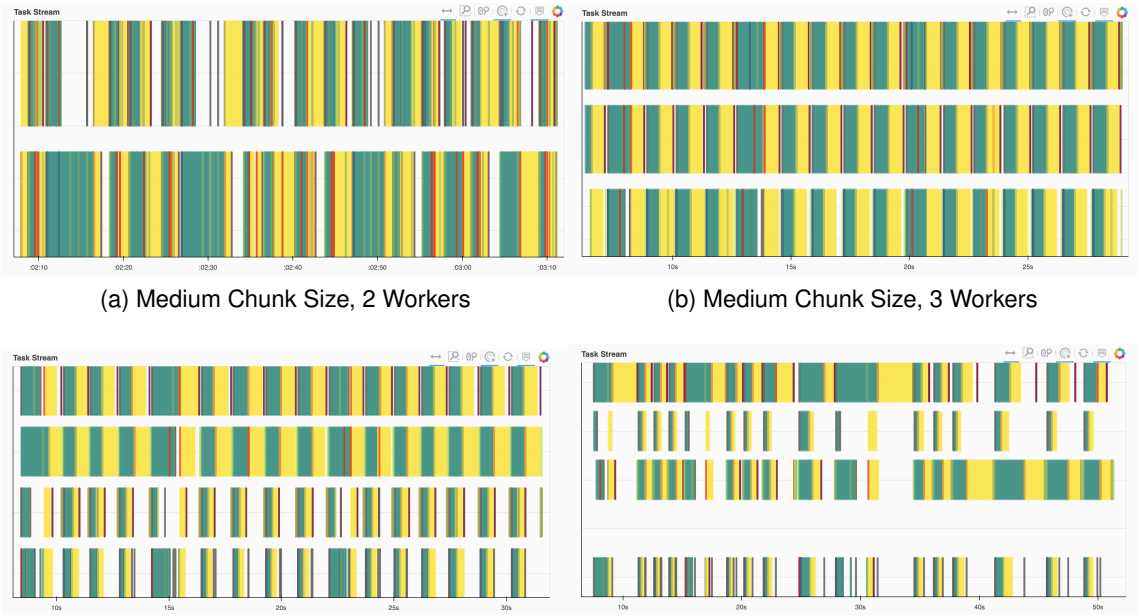
(a) Forward Stagewise Regression

(b) Least Angle Regression

Figure 9: Comparison of Speedup vs Number of workers (cores) between Forward Stagewise Regression and Least Angle regression with different chunk sizes

With just three workers, the speedup performance is increased by more than the factor of 2. After this point, the computations on the same number of NumPy arrays get divided among many workers, leading to faster computation on the chunks and more communication between the workers. Hence, there is not much improvement in the speedup. While more chunks lead to scheduler delay, fewer chunks also lead to less parallelization. This is verified by the slow increase in speedup with large chunk size in Figure 9(a). However, in the case of least angle regression, the trend is different from the forward stagewise regression. The peak speedup is achieved with a large chunk size and four workers. Medium chunk size means more communication between workers compared to large chunk size. As stated earlier, there are additional operations like matrix inversion and the calculation of parameter solution (γ) in the least angle regression that lead to more communication between workers. This produces overhead in the least angle re-

gression with medium chunk size. However, there is not much difference in the speedup with medium and large chunk size in Figure 9(b). Figure 10(b) shows the forward stagewise regression with three workers and medium chunk size, where we achieved the best speedup performance. There is almost no scheduler overhead and no communication between workers. Figure 11 showcases the forward stagewise regression with a large chunk size. We can see that the large chunk size leads to less parallelization.



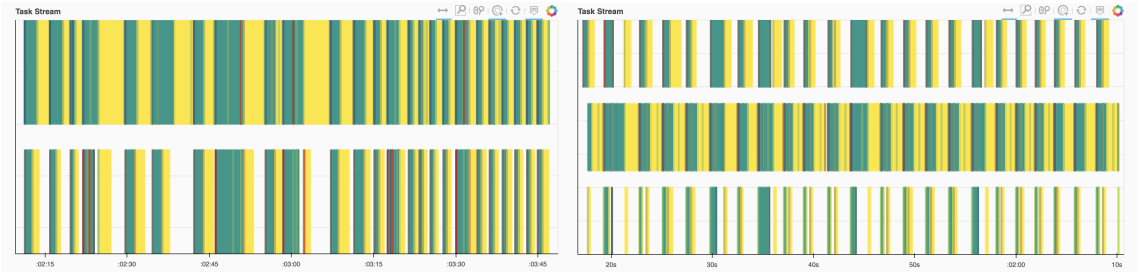
(a) Medium Chunk Size, 2 Workers

(b) Medium Chunk Size, 3 Workers

(c) Medium Chunk Size, 4 Workers

(d) Medium Chunk Size, 5 Workers

Figure 10: All figures correspond to forward stagewise with medium chunk size on a large dataset.

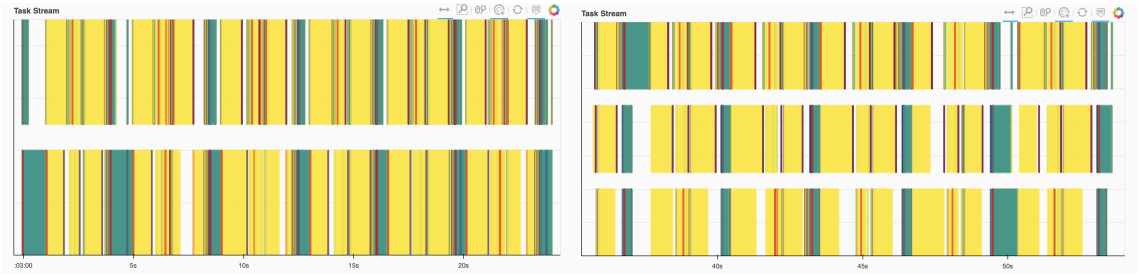


(a) Large Chunk Size, 2 Workers

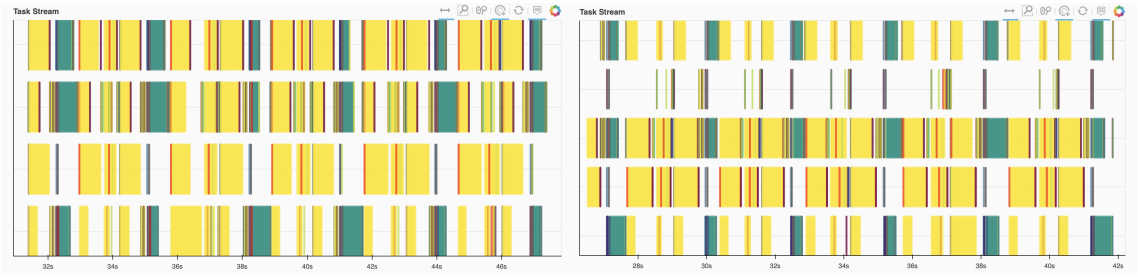
(b) Large Chunk Size, 3 Workers

Figure 11: The figures correspond to forward stagewise with large chunk size on large dataset.

Figures 12 and 13 show the activity of workers over time in the cluster for least angle regression with medium chunk size and large chunk size, respectively. As we can see in Figure 13(c), where we achieved the best performance for least angle regression, tasks are almost equally divided among workers, and there is very little or no communication between the workers (almost no red rectangles).

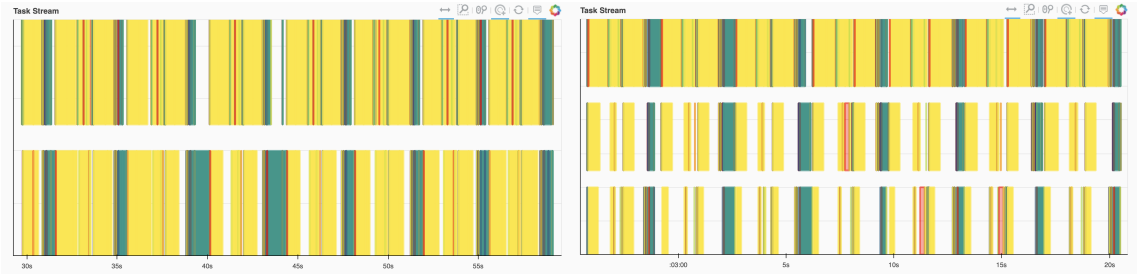


(a) Medium Chunk Size, 2 Workers (b) Medium Chunk Size, 3 Workers



(c) Medium Chunk Size, 4 Workers (d) Medium Chunk Size, 5 Workers

Figure 12: All figures correspond to least angle regression with medium chunk size on large dataset.



(a) Large Chunk Size, 2 Workers (b) Large Chunk Size, 3 Workers

(c) Large Chunk Size, 4 Workers (d) Large Chunk Size, 5 Workers

Figure 13: All figures correspond to least angle regression with large chunk size on large dataset.

5.3 End-to-End Comparison

The end-to-end comparison between forward stagewise and least angle regression for the first 500 variables on three different implementations is given in Table 3 and 4. The loading time of data into the cluster and the client's memory (for NumPy-multithreading)

Table 3: Forward Stagewise Regression with ($\epsilon = 0.05$) for the first 500 Variables on 3 different implementations

Implementation	No of Cores	Execution Time/s	No of iterations	Test Error
Sequential	1	5113.6	931	1.595
NumPy Multi-threading	40	571.25	931	1.595
Dask Distributed	**3**	**2045.2**	**931**	**1.595**

Table 4: Least Angle Regression for the first 500 Variables on 3 different implementations

Implementation	No of Cores	Execution Time/s	No of iterations	Test Error
Sequential	1	7284.2	500	1.587
NumPy Multi-threading	40	2070.8	500	1.587
Dask Distributed	**4**	**1937.3**	**500**	**1.587**

was excluded. The medium chunk size was used for forward stagewise, and the large chunk size was used for the least angle regression. In the case of forward stagewise regression, we achieved the speedup by more than the factor of 2 over sequential implementation with just three cores. However, NumPy multi-threading runs faster than our implementation. The only big calculation in the forward stagewise regression is the computation of the correlation vector where our large-scale input matrix **X** is used. Besides this, there are addition and subtraction operations on small vectors whose size is equal to the size of output vector **y**. As showcased by our results in section 5, the actual effect of distributed parallelization can only be seen on a large dataset. Therefore, NumPy is faster than our Dask distributed implementation. The actual effect of distributed parallelization is seen on our least angle regression implementation. We were able to get the speedup by the factor of 3.7 over sequential with just four cores. Our implementation is also faster than the NumPy Multi-threading that used all 40 cores of the machine. In the least angle regression, as more variables get added to the model, the array of active variables gets bigger to compute the parameter solution (γ). Therefore, when the calculation gets bigger, the distributed parallelization approach is cheaper and faster than the multi-threading approach. In terms of variables selected, both of the techniques produced the same set of significant variables from the dataset. The validation error is almost similar for both algorithms. However, the least angle regression runs faster than the forward stagewise regression for the first 500 variables on the Dask distributed implementation.

6 Conclusion and Future Work

This guided research implemented state-of-the-art variable selection approaches (forward stagewise regression and least angle regression) using Dask array and Dask distributed for parallel scalability. It compared both approaches in terms of variables selected and parallel scalability on the MD17 dataset.

The experimental findings show that the 100 variables are significant in the dataset. The generalization error of the techniques differs by 0.008 after 500 variables are in the model. This suggests that both methods included the same set of variables along the path. However, the learning rate made a considerable difference while training forward stagewise regression. Forward stagewise regression with a larger learning rate did not include a new variable after 1200 iterations. Therefore, a small learning rate should be used for the MD17 dataset to include all the variables. The variable selection techniques used in this research select the variable that has a substantial contribution to reducing the current residual. However, state-of-the-art performance for modeling quantum interactions in molecules was achieved by applying *continuous- filter convolutional layers* in SchNet [18]. The SchNet outperforms mean squared predictor on the MD17 dataset. Therefore, future work can be carried out to train SchNet with our selected set of variables and evaluate the generalization error.

As the parallel scalability of the algorithms, this thesis reports good results. We were able to achieve the speedup by more than the factor of 2 with just 3 CPU cores over sequential implementation for forward stagewise regression. For the first 50 iterations, there was no significant increase in the speedup performance in the case of the least angle regression. However, when more variables were added to the model, the speedup performance spiked up. The distributed implementation of the least angle regression was four times faster than the sequential implementation for the first 500 iterations. We were also able to beat the execution time of NumPy multi-threading with just four cores. Our results show that the least angle regression runs faster than the forward stagewise regression in the distributed setting. However, recording the execution time for just 500 iterations is not enough when analyzing the parallel scalability of the algorithms. The chunk size plays an essential role in the parallelization with Dask. In our work, we evaluated the speedup performance with respect to three chunks sizes. Although we achieved good speedup performance, running the algorithms for several other chunk sizes could give us better speedup results. In addition to that, the dataset used in this research was not significantly large to analyze the advantages and disadvantages of distributed parallelization. Therefore, there is possible future research that can be carried out in this domain.

The future work that we are proposing can be summarized as 1) Train the SchNet with our selected variable and evaluate the prediction performance 2) Run the algorithms for longer iterations and analyze speedup performance 3) Run the algorithms with several chunk sizes to find the optimal chunk size 4) Train the model using the larger dataset to get a clear idea of parallel scalability.

References

[1] Stefan Chmiela et al. “Machine Learning of Accurate Energy-Conserving Molecular Force Fields”. In: *Science Advances* 3 (May 2017), e1603015. DOI: 10.1126/sciadv.1603015.

[2] Stefan Chmiela et al. “Towards Exact Molecular Dynamics Simulations with Machine-Learned Force Fields”. In: *Nature Communications* 9 (Feb. 2018). DOI: 10.1038/s41467-018-06169-2.

[3] Dask. *Scalable analytics in Python*. https://dask.org. Accessed: 2020-05-15.

[4] DeepMind. *AlphaGo*. https://deepmind.com/research/case-studies/alphago-the-story-so-far. Accessed: 2020-05-15.

[5] John Ehrlinger. “Regularization: Stagewise Regression and Bagging”. PhD thesis. Case Western Reserve University, 2011.

[6] Hongyu Guo. “Generating Text with Deep Reinforcement Learning”. In: *CoRR* abs/1510.09202 (2015). arXiv: 1510.09202. URL: http://arxiv.org/abs/1510.09202.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[8] Bing Huang and O. Anatole von Lilienfeld. *Efficient accurate scalable and transferable quantum machine learning with am-ons*. 2017. arXiv: 1707.04146 [physics.chem-ph].

[9] IBM. *A Computer Called Watson*. https://www.ibm.com/ibm/history/ibm100/us/en/icons/watson/. Accessed: 2020-05-15.

[10] IBM. *Deep Blue*. https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/. Accessed: 2020-05-15.

[11] Alexandra L’Heureux et al. “Machine Learning with Big Data: Challenges and Approaches”. In: *IEEE Access* PP (Apr. 2017), pp. 1–1. DOI: 10.1109/ACCESS.2017.2696365.

[12] Guillaume Lample et al. “Phrase-Based & Neural Unsupervised Machine Translation”. In: *CoRR* abs/1804.07755 (2018). arXiv: 1804.07755. URL: http://arxiv.org/abs/1804.07755.

[13] Quoc Le et al. “Building high-level features using large scale unsupervised learning”. In: *Proceedings of ICML* 1 (Dec. 2011).

[14] Chih-Hao Lo, Jingxia Yuan, and Jun Ni. “Optimal temperature variable selection by grouping approach for thermal error modeling and compensation”. In: *International Journal of Machine Tools and Manufacture* 39 (Sept. 1999), pp. 1383–1396. DOI: 10.1016/S0890-6955(99)00009-7.

[15] Sebastián Maldonado and Richard Weber. “Weber, R.: A wrapper method for feature selection using support vector machines. Inf. Sci. 179(13), 2208-2217”. In: *Inf. Sci.* 179 (June 2009), pp. 2208–2217. DOI: 10.1016/j.ins.2009.02.014.

[16] Sharon Qian and Yaron Singer. “Fast Parallel Algorithms for Feature Selection”. In: *ArXiv* abs/1903.02656 (2019).

[17] Kristof Schütt et al. “Quantum-Chemical Insights from Deep Tensor Neural Networks”. In: *Nature Communications* 8 (Jan. 2017). DOI: 10.1038/ncomms13890.

[18] Kristof Schütt et al. “SchNet: A continuous-filter convolutional neural network for modeling quantum interactions”. In: June 2017.

[19] J. Suto, S. Oniga, and P. P. Sitar. “Comparison of wrapper and filter feature selection algorithms on human activity recognition”. In: *2016 6th International Conference on Computers Communications and Control (ICCCC)*. 2016, pp. 124–129.

[20] Robert Tibshirani et al. “Least angle regression”. In: *The Annals of Statistics* 32.2 (Apr. 2004), pp. 407–499. ISSN: 0090-5364. DOI: 10.1214/009053604000000067. URL: http://dx.doi.org/10.1214/009053604000000067.

[21] Ioannis Tsamardinos et al. “Massively-Parallel Feature Selection for Big Data”. In: *ArXiv* abs/1708.07178 (2017).

[22] Haohan Wang, Bhiksha Raj, and Eric P. Xing. “On the Origin of Deep Learning”. In: *ArXiv* abs/1702.07800 (2017).

[23] Jonathan Wang et al. “Parallel Variable Selection for Effective Performance Prediction”. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2017), pp. 208–217.

[24] Lei Yu and Huan Liu. “Efficient Feature Selection via Analysis of Relevance and Redundancy”. In: *J. Mach. Learn. Res.* 5 (Dec. 2004), pp. 1205–1224. ISSN: 1532-4435.

[25] Emilia Zawadzka-Gosk, Krzysztof Wołk, and Wojciech Czarnowski. “Deep Learning in State-of-the-Art Image Classification Exceeding 99% Accuracy”. In: Apr. 2019, pp. 946–957. ISBN: 978-3-030-16180-4. DOI: 10.1007/978-3-030-16181-1_89.