MACHINE LEARNING MODEL SELECTION FOR PREDICTING
PROPERTIES OF HIGH REFRACTIVE INDEX POLYMERS

by

Gaurav Vishwakarma

September 2018

A thesis submitted to the
Faculty of the Graduate School of
the University at Buffalo, State University of New York
in partial fulfilment of the requirements for the
degree of

Master of Science

Department of Chemical and Biological Engineering
The thesis of Gaurav Vishwakarma was reviewed by the following:

Johannes Hachmannn
Assistant Professor of Chemical and Biological Engineering
Thesis Advisor, Chair of Committee

David A. Kofke
SUNY Distinguished Professor of Chemical and Biological Engineering
Committee Member
I am deeply grateful to my advisor, Prof. Johannes Hachmann, for his continuous support, patience, and encouragement. I could not have imagined having a better advisor to look up to for my Masters degree. I would like to thank Prof. David Kofke for serving on my committee, and for his insightful comments and encouragement.

Next, I would like to take a moment to acknowledge Mojtaba Haghighatlar for being there as a mentor and friend, and guiding and supporting me throughout my thesis. During the course of my stay in Buffalo, I was fortunate to work with a number of remarkable colleagues and friends: Mohammad Atif Faiz Afzal, Amol R. Mahajan, Christine LaPorte, Aditya Sonpal, PoHan Chen, and other members of Hachmann research group. They have all made my time in Buffalo cheerful and rewarding.

Finally, I would like to thank my parents, Ganesh Vishwakarma and Aruna Vishwakarma and also my sister, for their constant love, support and guidance throughout my life.
# Table of Contents

Acknowledgments iii

Abstract vii

Chapter 1  
Introduction 1  
1.1 Background ................................................. 1  
1.1.1 Machine Learning for Materials Chemistry .......... 1  
1.1.2 Model Selection ......................................... 2  
1.1.3 Data Driven Research in Our Group .................. 3  
1.2 Hyper-parameter Optimization ............................ 5

Chapter 2  
Search Algorithms for Hyper-parameter Selection 8  
2.1 Grid Search .................................................. 8  
2.2 Random Search .............................................. 9  
2.3 Particle Swarm Optimization .............................. 10  
2.4 Genetic Algorithm ......................................... 12

Chapter 3  
General Methodology 15  
3.1 Schematic .................................................. 15  
3.2 Data Set .................................................... 16  
3.3 Model Hyper-parameters ................................. 17  
3.4 Evaluation Metric ......................................... 18

Chapter 4  
Results and Discussions 20
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Grid Search</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Random Search</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Particle Swarm Optimization</td>
<td>22</td>
</tr>
<tr>
<td>4.4 Genetic Algorithm</td>
<td>23</td>
</tr>
<tr>
<td><strong>Chapter 5</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary and Outlook</strong></td>
<td>28</td>
</tr>
<tr>
<td>5.1 Conclusions</td>
<td>28</td>
</tr>
<tr>
<td>5.2 Challenges to Hyper-parameter Optimization</td>
<td>29</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>30</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Random Search ........................................ 21
4.2 Particle Swarm Optimization .......................... 22
4.3 Genetic Algorithm ....................................... 23
4.4 Genetic Algorithm reduced data set .................. 26
4.5 Feature Selection ....................................... 27

List of Figures

1.1 Model Selection ......................................... 3
1.2 Hachmann group’s software ecosystem ............... 4
2.1 Genetic Algorithm ....................................... 12
2.2 Genetic Algorithm Operators ........................... 13
3.1 Hyper-parameter Selection scheme ..................... 15
4.1 Learning Curve ......................................... 25
Abstract

In the field of materials science and chemistry, machine learning has emerged as a promising technique in the recent times for the accelerated discovery of novel materials. This thesis focuses on one of the major aspects of machine learning, i.e., model selection, which is an important and also time-intensive task but remains highly unexplored in the materials community. We present a framework for automated model selection for machine learning with our research group’s current work in the prediction of properties of organic polymers as the primary focus.

The traditional approach for hyper-parameter optimization of a given machine learning model is discussed in the beginning, followed by the need for more specialized techniques due to the shortcomings of this approach that are readily identified. We analyze two algorithms for hyper-parameter selection: genetic algorithm and particle swarm optimization. For this purpose, we develop a genetic algorithm and particle swarm optimization module which is then incorporated within our research group’s machine learning software pack-
The algorithms are compared based on their performance as well as the time taken for the optimization to complete. It is shown that both genetic algorithm and particle swarm optimization are able to find better hyper-parameter values compared to the traditional methods used for hyper-parameter tuning, but at the cost of slightly higher computational time. Two approaches for reducing the computational time are also explored; one approach being feature selection using genetic algorithm, while the other is reducing the size of the data. It is shown that both the methods result in a lower computational time without losing much on the prediction accuracy of the machine learning model.
Chapter 1

Introduction

1.1 Background

1.1.1 Machine Learning for Materials Chemistry

Materials research has traditionally been guided by laboratory experiments and in the past few decades by computational modelling and simulation. However, the discovery of novel materials has been remarkably slow. Experiments in the lab have widely been performed based on intuition, and are both time and resource intensive. Computational experiments too suffer from similar drawbacks of computational time and accuracy based on level of theory. With the advances in computer hardware and algorithms in the past few decades, computational simulation and modelling have seen a transformation. More and more data is being generated every day by the use of high-performance computing clusters. Over the recent years, there has been a steady shift in the approach towards materials research [1] [2], moving from pure method development techniques to
machine learning [3] [4] [5] [6] which is being used for analyzing the data being generated for the accelerated discovery of novel materials [7].

Machine learning [8] [9] is a powerful tool derived from pattern recognition techniques where the algorithm can ‘learn’ from the data and is able to make predictions on the data that are fast and accurate. A machine learning model tries to fit a function when mapping the features to the target labels or values. Better the fit, better is the performance of the model.

Machine learning tasks are typically categorized into Supervised learning and Unsupervised learning [10]. Supervised learning is a class of machine learning where the algorithm learns the patterns in data through sample input-output pairs and is then able to make predictions on ‘unseen’ data. Unsupervised learning on the other hand applies to unlabeled data or data that does not have sample input-output pairs but only has a set of inputs. The algorithm in this case tries to identify relationships between the inputs and the inputs are then commonly put into groups or ‘clusters’.

Based on the output, machine learning tasks can also be categorized as classification, regression and clustering. The first two are examples of supervised learning, while clustering falls into the category of unsupervised learning. Classification refers to categorizing the outputs into different labelled classes or sections, and when the outputs can assume continuous values, it is referred to as regression analysis.

1.1.2 Model Selection

When building a machine learning model, the most critical part of the process is selecting the type of algorithm/model that will be used to fit the data. Typi-
cally, this is done by considering the size of input data and the type of prediction being done by the model. For example in regression analysis in machine learning, neural networks are commonly used for modelling large and complex non-linear data.

As shown in Fig 1.1, to construct a machine learning model, there are two required inputs. The first being the data that is fed to the algorithm so that it can ‘learn’ the pattern in the data. The second set of inputs are known as ‘hyper-parameters’, which are the initial inputs that need to be given to the model before the learning process actually starts. Hyper-parameters govern the behaviour of the learning process itself, and are different from ‘model parameters’ which are the parameters that the algorithm tunes during the learning process in order to get a better fit of the data. With these two inputs, the algorithm starts learning the data and its performance is measured by a given metric.

1.1.3 Data Driven Research in Our Group

In our group, we have been developing a software-ecosystem for data-driven design of chemical systems and exploration of chemical space [11]. The software ecosystem is composed of four different packages:
1. ChemLG: Molecular Library Generator

2. ChemHTPS: Virtual Highthroughput Screening Infrastructure

3. ChemBDDB: Database Infrastructure

4. ChemML: Data Analysis, Mining and Modelling Infrastructure

The group generates data through the ChemHTPS infrastructure and data mining is performed via the ChemML suite [12]. The ChemML suite includes data analysis, mining and modelling capabilities that allow us to apply state-of-the-art machine learning and informatics methodology to chemical and materials data sets. For our group to perform data analysis on any given data set, there is
a need for a deeper insight into model selection.

1.2 Hyper-parameter Optimization

As mentioned in Sec. 1.1.2, a machine learning model takes two inputs for it to be able to make predictions accurately; a data set and a set of hyper-parameters. Given a data set, the next step is to decide upon the algorithm to be used for machine learning. Barring a few, almost all algorithms come with a set of hyper-parameters that have a significant impact on the prediction accuracy of the model. Finding the right set of hyper-parameters is therefore crucial to the performance of a machine learning model in predicting a target property. However, no single set of hyper-parameters for a machine learning model can be generalized for all data sets.

The hyper-parameter values can be of discrete as well as continuous nature. Discrete values can be explored exhaustively at the cost of high computational time but the same cannot be done for continuous search space. The complexity of the problem is exacerbated when the hyper-parameters of interest start increasing in number.

Traditionally, the hyper-parameters for a model have been decided based on intuition and/or experience. Of late, use of trivial search algorithms such as Grid Search and Random Search has seen a rising trend for tuning of hyper-parameters. Bergstra et al. [13] have shown Random Search to outperform Grid Search in terms of better models and lower computation time. Iterative gradient descent methods have also been suggested for tuning of hyper-parameters with a major assumption that the hyper-parameter search space is differentiable throughout [14]. Even though there exist a large number of search algorithms
that can be used for tuning of hyper-parameters [15], the use of other search algorithms, apart from those mentioned above, is extremely limited because these calculations tend to be very expensive in terms of computational time. Bayesian Optimization [16] has been shown to be a promising candidate among search algorithms but is limited by its sequential nature which inhibits it from being parallelized on high performance computing clusters.

The materials informatics community is faced with a persistent question about the necessity for specialized search algorithms for hyper-parameter tuning. This study is aimed at answering this question and in the process identifying a strategy for tuning of hyper-parameters involved in training a machine learning model. The expectation is that the proposed strategy results in a substantial improvement in the accuracy of the machine learning model while not compromising too much on time. For this purpose, two evolutionary search algorithms are studied and employed for tuning of hyper-parameters and are compared based on their prediction accuracy and efficiency. Furthermore, the proposed strategy is to be implemented as a module in the ChemML suite which can be used for tuning the hyper-parameters of any given machine learning model.

The effect of hyper-parameter optimization is studied on the data set from an on-going project [17] [18] in our group that deals with the prediction of properties such as refractive index, density and polarizability of organic polymers. The dataset comprises of 100,000 molecules, compiled by performing first-principles calculations and molecular dynamics simulations to calculate the aforementioned properties. For extending these calculations to 1.5 million similar molecules, an efficient way to reduce the computational time was to use ChemML to train a machine learning model for predicting their properties as
opposed to actually performing the expensive molecular dynamics simulations for this task.

This thesis is organized as follows: Chapter 2 describes in detail the four search algorithms that are used for hyper-parameter optimization in this thesis. Chapter 3 deals with the methodology and data sets used for conducting the computational experiment. Results and Discussion are presented in Chapter 4 followed by the Summary and Outlook in Chapter 5.
Chapter 2

Search Algorithms for
Hyper-parameter Selection

2.1 Grid Search

One of the most basic and widely used algorithms for finding the optimum set of hyper-parameters is Grid Search. Grid Search essentially involves performing an exhaustive search over all possible combinations of the hyper-parameters that are manually provided to the algorithm. For low dimensionality problems, i.e., where the possible combinations of the hyper-parameters is very low, this is indeed a very good tactic.

However, an obvious drawback of the method is that since some hyper-parameters can take on real values as well, some kind of discretization, usually guided by intuition, is also necessary in addition to manually setting the bounds of the corresponding hyper-parameters. Additionally, if a model is highly sensitive to a certain real-valued hyper-parameter, the discretization should be done very finely over the entire valid range of the hyper-parameter, which will result
in a manifold increase in the possible combinations that need to be exhausted by the algorithm. Similarly, as the dimensionality of the search problem increases, the volume of the search space increases extremely fast and it becomes highly inconvenient to use grid search in such problems owing to the time factor.

2.2 Random Search

Random Search refers to the random selection of a combination of hyper-parameter values in the valid search space and testing them using the specified performance metric. For a hyper-parameter search space with a relatively low dimensionality, this proves to be an efficient method as compared to Grid Search. It also overcomes the issue of discretization of real-valued hyper-parameter search space that is required in Grid Search as values are picked randomly from the specified range.

However, the random search method still doesn’t address the issue of sampling the valid search space for a model with high sensitivity to one or more real-valued hyper-parameters. In order to properly sample the search space via random search, the required number of iterations goes up and the problem is exacerbated as the number of real-valued hyper-parameters increases.

Both Grid Search and Random Search are limited in their performance since the information gathered while testing of each hyper-parameter combination is not used in the future trials. This is where ‘smart’ search algorithms outperform the traditional methods of selecting hyper-parameters for a machine learning model.
2.3 Particle Swarm Optimization

Particle Swarm Optimization [19] [20] falls under the category of Swarm Intelligence methods. Swarm Intelligence methods represent the collective behavior of decentralized self-organized systems. Members of a swarm intelligence are guided by local interactions with one another and the environment. Like Genetic Algorithm, Particle Swarm Optimization is also inspired by biological systems such as bird flocking or a school of fish.

The algorithm resembles the pattern seen in birds while searching for a food source. All the birds in a flock act independently and search for food at the same time while communicating with each other locally the location of the food if they find any. Without the presence of a central control, the birds can either keep going in their respective directions or can choose to follow the direction of the bird that reported being closest to the food source.

The algorithm follows the following general steps. A population (swarm) of candidate solutions (particles) is instantiated. Each particle is assigned a random initial position and initial velocity. Fitness values of all particles in the swarm are calculated by evaluating the objective function for each particle. The particles are then ‘flown’ through the search space.

This movement is guided by two major factors: a particle’s own best position (also known as the cognitive parameter, c1) and the swarm’s overall best position (also known as the social parameter, c2). The particles can be biased toward one of these two factors by adjusting the weight (w) of the particle. A heavier particle will have a larger inertia and will resist a change in its momentum (or velocity vector) which makes it more likely to follow its own best position while ignoring the swarm’s overall best position. With a similar reasoning, a lighter
particle will be more inclined to follow the swarm’s overall best position. The equations guiding the movements of the particles are:

\[
cognitive = (c_1 \times \text{rand}(0, 1) \times (p_{\text{best}} - p_i)) \tag{2.1}
\]

\[
social = (c_2 \times \text{rand}(0, 1) \times (g_{\text{best}} - p_i)) \tag{2.2}
\]

\[
v_{i,\text{new}} = (w \times v_i) + cognitive + social \tag{2.3}
\]

\[
p_{i,\text{new}} = p_i + v_{i,\text{new}} \tag{2.4}
\]

where for the \(i^{th}\) particle,

\(p_i\) is its current position,

\(p_{\text{best}}\) is the particle’s personal best position,

\(g_{\text{best}}\) is the swarm’s best known position,

\(v_i\) is the current velocity of the \(i^{th}\) particle,

and \(v_{\text{new}}\) and \(p_{\text{new}}\) are the new velocity and position of the \(i^{th}\) particle respectively.

After the particles are moved around in search space, new fitness values are calculated for each particle and the process repeats until a stopping criteria is met. The algorithm is known to perform well on real-valued search space [21] in contrast to Genetic Algorithm (Sec 2.4) which has historically been used for solving binary coded problems. However, a delicate balance has to be struck between the social and cognitive parameter so as to cover the search space efficiently and at the same time avoiding getting stuck in a local minima [22].

The python implementation of Particle Swarm Optimization is obtained from the public repository Pyswarms [23], on Github.
2.4 Genetic Algorithm

A subclass of Evolutionary Algorithms (EA) which are inspired by biological evolution, Genetic Algorithm (GA) [24] [25] is used for solving both constrained and unconstrained search and optimization problems. The algorithm (Fig 2.1) follows the rules of mutation, crossover and selection, similar to the ones seen in biological evolution.

GA starts with an initial ‘population’ of candidate solutions for the optimization problem, also called ‘chromosomes’ or ‘individuals’. Every chromosome is represented as a linear array of binary or real-valued numbers. The array is composed of elements called ‘bits’ or ‘genes’, with each bit containing information that affects the overall behavior of the chromosome. The bits in a chromosome could traditionally assume only binary values, but the use of real-coded bits in a chromosome is also not uncommon.

![Figure 2.1: Genetic Algorithm](image)

The performance of the individuals is measured by evaluating a given cost
function that returns a metric for comparison [26]. The initial population is evaluated based on the cost function. A ‘selection’ process is used to select a subset of individuals for ‘breeding’, which means performing the mutation and crossover operators; and evaluating the transformed individuals using the cost function. The crossover operator (Fig 2.2a) is responsible for combining the ‘genetic’ information that is contained in the parents and transferring them to the ‘offspring’. This is achieved computationally by swapping some of the bits from the two individuals. The mutation operator (Fig 2.2b) introduces diversity or randomness in a chromosome. One or more bits of a chromosome are picked and assigned values randomly within the allowable range.

A selection process is again applied to select individuals which will serve as ‘parents’ for the next ‘generation’ of individuals. There are a number of methods for selection that differ mainly in the choice of candidates that should be available in the next generation. Some methods are biased toward the better performing individuals while others try to include the ones that do not perform well with the hope that the chromosome could still contain certain genes that could enhance the performance of the individual. After several iterations are performed, the ‘good’ genes survive through the generations and the population is said to ‘evolve’ toward a better solution.

![Figure 2.2: Genetic Algorithm Operators](image)

Within the *ChemML* suite, genetic algorithm is implemented based on
the Distributed Evolutionary Algorithms in Python (DEAP) library [27]. The ChemML genetic algorithm has four built-in methods that differ in the ‘selection’ operation.
General Methodology

3.1 Schematic

For hyper-parameter selection, all the four algorithms search algorithms are set up following the scheme shown in Fig 3.1.

![Diagram](image)

Figure 3.1: Hyper-parameter Selection scheme

The data is first split into a common ‘test set’ and ‘training set’ in a 1:9 ra-
tio. The training set is used to train machine learning models for the candidates generated by the search algorithms. The performance of the candidates is measured by the mean absolute error of the corresponding model. For calculation of the mean absolute error, the model is first trained on the training set; the trained model is then used to predict a small subset of the training data; and the difference between the predicted and actual values averaged over all predictions is the mean absolute error.

In order to better represent the error of the model, a five-fold cross validation is done so that every data point in the training set gets a chance to be in the test set. The model with the least mean absolute error (MAE) is selected and the corresponding hyper-parameters are used to train the model on the entire training set. The trained model is used to predict the ‘unseen’ test set that was split at the beginning of the search. The MAE of the prediction is the final error used for comparing the different search algorithms.

## 3.2 Data Set

The dataset for the machine learning model is from our group’s current work in the prediction of properties (Refractive Index, Polarizability and Density) of 100,000 organic polymers. The molecular descriptors for these molecules form the ‘training data’, while the Refractive Index, Polarizability and Density form the ‘training labels’ (Fig 1.1).

Molecular descriptors contain chemical information represented in terms of numerical values that are obtained by performing mathematical operations on the molecule. Previous work by our group concludes that features or ‘descriptors’ generated from the ‘Dragon’ application are ideal for this data set. Of a
total of 5270 molecular descriptors provided by Dragon, only 2D descriptors (1893) are selected for the 100,000 molecules, which makes the dataset for machine learning to be of the size of $100000 \times 1893$, out of which 90000 data points are used for training while the remaining 10000 is used for testing. The data set handling in python is done via the 'Pandas' library.

### 3.3 Model Hyper-parameters

Neural networks are a good candidate for the model used for fitting in this case primarily due to the size of the data set. Neural networks also provide more flexibility while fitting a model and are ideal candidates for studying hyper-parameter optimization since they have a larger number of hyper-parameters that require tuning.

**Neural networks** or more specifically Artificial Neural Networks (ANNs), are supervised machine learning algorithms which are inspired by biological neural networks. Similar to the neurons in a biological brain, an ANN is comprised of artificial neurons that are connected to each other with the only difference being that the artificial neurons are organized in multiple layers. The artificial neurons, henceforth referred to as neurons, communicate with each other via signals (many-to-many interaction) that resemble interacting neurons in an actual brain.

A neuron receives multiple signals (real numbers), and applies an activation function (linear or non-linear) to the cumulative input that it receives. Every input signal in the ANN is also associated with an initial weight. These weights, which decide the strength of the signal transmitted by the corresponding neurons, are updated with the introduction of every data point. The modified signal
is then passed on to the next layer of neurons. The first layer in an ANN is the input layer that receives the input data and has a number of neurons equal to the number of features in the input data. The final layer in an ANN is called the output layer, with the number of neurons equal to the number of target properties being predicted (usually just one), whose output is the final output of the ANN. The intermediate layers in an ANN are called ‘hidden’ layers.

The number of hidden layers along with the number of neurons per hidden layer, the type of activation function and the regularization parameter (which regulates over-fitting in a model) are the hyper-parameters of interest in this study [28] [29]. The range or allowable values of these hyper-parameters are:

- Number of hidden layers: 1 / 2 / 3
- Number of neurons per hidden layer: (0, 250)
- Activation function: Identity / logistic / tanh / relu
- Regularization parameter: (0.00001, 1)

Since the valid range of the number of neurons (0-250) is quite big, it is first discretized before using any algorithm. The discretization is done by dividing the valid range in steps of 20 starting from 0. The python implementation of neural networks is used from the ‘Scikit-Learn’ library.

### 3.4 Evaluation Metric

For hyper-parameter selection, the machine learning libraries are obtained from the ChemML suite. For the purpose of comparison, the final model from each search algorithm is tested for accuracy on the same ‘test set’ (Fig 3.1) which
is a subset of the initial dataset. Neural networks are used for mapping the descriptors to all the three target properties or ‘labels’. Previous work by our group also suggests that a ‘many to many’ mapping for all the three properties gives similar errors when compared to a ‘many to one’ mapping for each of the target labels individually. Hence, the neural networks are designed to predict the three properties simultaneously. However, only the MAE of Refractive Index is used in the cost function.
Results and Discussions

The computational time reported for all the algorithms below are for the following CPU specifications:
Number of nodes: 1
Number of cores per node: 16
Node memory: 64 GB
Jobs are run parallel on the 16 core node via the Scikit-Learn package.

4.1 Grid Search

For grid search, if 10 values of the regularization parameter are considered in the range (0.00001, 1), then the total number of hyper-parameter combinations to be evaluated are:

\[
\text{regularization parameter (10)} \times \text{activation function (4)} \times \text{number of neurons for 3 hidden layers (12^3)} = 69120 \text{ combinations.}
\]

Even with such a small number of discretized parameter values, the total
number of hyper-parameter combinations to be evaluated is extremely high. The time taken for one neural network model to be trained with 90k data points running parallel on a 16-cores node is approximately 6-7 minutes. At this rate, it would take approximately one year to complete the grid search for the 69k combinations.

Since the regularization parameter has a real valued search space, it would take a much larger set of values to be able to scan a good majority of the search space. Hence, grid search is not a good choice for the model selection problem presented here.

4.2 Random Search

The random search method is given a range of possible hyper-parameter values (Sec. 3.3) and is run for 25 iterations. The best results after 10 random search runs are summarised in Table 4.1.

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>MAE Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regularization parameter</td>
<td>8.59E-03</td>
</tr>
<tr>
<td>Activation function</td>
<td>Polarizability</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>2.874</td>
</tr>
<tr>
<td></td>
<td>Density</td>
</tr>
</tbody>
</table>

When compared to the approximate time required for grid search, random search is completed in a very small time of just around 3 hours for the 25 iterations, which is a total of 250 evaluations of the machine learning model.
4.3 Particle Swarm Optimization

The parameter values used in the algorithm are cognitive=2, social=2, inertia=0.01, velocity clamp=(0,1). Since the algorithm is known to perform well on real-valued search space, the optimization is first done only for the regularization parameter for 25 iterations with a swarm size of 20, and the results are shown in Table 4.2.

Table 4.2: Particle Swarm Optimization

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regularization parameter</td>
<td>5.8E-04</td>
</tr>
<tr>
<td>Activation function</td>
<td>relu</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>128, 64, 32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAE Test Set</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refractive Index</td>
<td>7.4E-03</td>
</tr>
<tr>
<td>Polarizability</td>
<td>2.25</td>
</tr>
<tr>
<td>Density</td>
<td>5.471</td>
</tr>
</tbody>
</table>

Considering only the MAE of the test set for Refractive Index, the algorithm has a 14% improvement over random search. Predictions for polarizability and density also improve by 21.7% and 27.9% respectively. Although the algorithm performed better than random search, the time required for particle swarm optimization to reach the optimal solution was a little over 4 days. We believe this is because of the need to update the neighborhood matrix for each particle after every iteration. Moreover, the time taken for the optimization will definitely go up as more dimensions (the remaining hyper-parameters) are introduced in the cost function.
4.4 Genetic Algorithm

The optimization results of genetic algorithm for 25 generations with a population size of 30 are tabulated in Table 4.3. There is a little more than 15% improvement in the MAE of refractive index, and around 18.5% and 30% improvement in the prediction accuracy for the polarizability and density respectively, when compared to random search. The genetic algorithm predictions are also slightly better than those of particle swarm optimization.

Table 4.3: Genetic Algorithm

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regularization parameter</td>
<td>1.38E-04</td>
</tr>
<tr>
<td>Activation function</td>
<td>relu</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>120, 240, 160</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAE Test Set</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refractive Index</td>
<td>7.28E-03</td>
</tr>
<tr>
<td>Polarizability</td>
<td>2.344</td>
</tr>
<tr>
<td>Density</td>
<td>5.372</td>
</tr>
</tbody>
</table>

It took Genetic Algorithm close to a day and a half to arrive at these results, which is almost one-third of the time taken by particle swarm optimization. For the dataset used here (100000*1893), this time is comparable to that of 10 runs of random search. But for the case of larger datasets, in order to reduce the time taken by Genetic Algorithm, there can be a number of ways to do just that.

1. **Reducing the number of generations of Genetic Algorithm.** Although it seems like a good idea at first, reducing the iterations of the algorithm is not advisable as it can result in non-convergence of the algorithm.

2. **Reducing the number of training data points fed to the neural network.** This is a bit tricky as reducing the number of data points to the neural
network usually results in poor prediction accuracy. Recent studies in machine learning have shown that some data points contribute more to the training of the model as compared to others. And, as a general rule of thumb, more the diversity in the training data points fed to the algorithm, better is the prediction of that model for new unseen data.

However, even if the prediction accuracy were to be ignored for a moment and the hyper-parameter tuning was done for the reduced data set, the best hyper-parameter set resulting from this experiment might not be the best hyper-parameter set for the original data set. There also exists an uncertainty in the use of this method since it is highly-dependent on the kind of data set fed to the algorithm.

3. **Reducing the feature space of the data.** This method falls under the scope of ‘dimensionality reduction’. The idea behind this approach is that not all descriptors are considered to contribute equally to the model. Feature selection is one of the ways of performing dimensionality reduction by selecting the ‘most important’ features for representing the data.

Reiterating the statement made in the previous method, reducing the data fed to the algorithm is usually expected to degrade the performance of the model. However, when the set of descriptors used for representation of the molecules gets larger, it becomes necessary to perform feature selection to be able to perform any kind of data mining on that data.

The latter two methods mentioned above are implemented using Genetic algorithm and the results are summarised below.

The effect of **reducing the data points** on the cross-validated MAE is seen in Fig 4.1. The MAE curve starts to flatten out near 40k data points, and at 60k
data points the MAE almost equals that at 90k data points. So, hyper-parameter optimization is performed for 40k and 60k data points selected randomly from the original data set, and the results are tabulated in Table 4.4.

![Learning Curve](image)

Figure 4.1: Learning Curve

Although the 40k data points have a very similar MAE for test set prediction as that of the original data set, there is a big disparity in the predictions for 60k data points. Also, the 40k data points have a lower MAE than that for 60k data points as well as a lower computational time of 17 hours as compared to 28 hours for the 60k data points.

These results show that a model can indeed be trained with a reduced data set without losing too much on accuracy. The trick is to decide upon the number of data points to use for training. It should also be noted that in this case, the 40k and 60k values were chosen based on visual examination of the learning
Table 4.4: Genetic Algorithm reduced data set

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>Data points</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>40k</td>
</tr>
<tr>
<td>Regularization parameter</td>
<td>1.778E-03</td>
</tr>
<tr>
<td>Activation function</td>
<td>relu</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>80, 180, 240</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAE Test Set</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Refractive Index</td>
<td>7.29E-03</td>
<td>8.09E-03</td>
</tr>
<tr>
<td>Polarizability</td>
<td>2.367</td>
<td>2.557</td>
</tr>
<tr>
<td>Density</td>
<td>5.82</td>
<td>6.17</td>
</tr>
</tbody>
</table>

curve alone.

However, these results are somewhat contrary to the expectation. Increasing the number of data points in the training set usually increases the prediction accuracy of the model. So, one would expect the model trained on 60k data points to perform better than the model trained on 40k data points. However, that is not the case here. This behaviour can be explained based on the 10k data points selected for testing. If a machine learning model is trained on data points that resemble the data points in the test set, the model should be able to predict those data points with high accuracy. So, it might be the case that the 40k data points have more resemblance to the test set data than the 60k data points.

Next, feature selection was done using genetic algorithm with a population size of 30 and number of generations as 30. The cost function for feature selection was also based on the cross-validated mean absolute error of the training set, which drives the genetic algorithm towards a lesser number of descriptors and at the same time a smaller mean absolute error.

The genetic algorithm converged to a total of 818 features out of a total of 1893 original features. The results of the genetic algorithm post feature selec-
tion for hyper-parameter optimization are shown in Table 4.5. These values are very similar to those from the original data set in Table 4.3, which suggests that a machine learning model can be built by using lesser number of features/descriptors with almost the same accuracy. An important point to note here is that the best set of hyper-parameters obtained after feature selection has only two hidden layers with the number of neurons for each hidden layer as 240. This means that the machine learning model can now be trained in less time as there are fewer parameters for the neural network to tune.

Table 4.5: Feature Selection

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regularization parameter</td>
<td>1.22E-04</td>
</tr>
<tr>
<td>Activation function</td>
<td>relu</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>240, 0, 240</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAE Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refractive Index</td>
</tr>
<tr>
<td>Polarizability</td>
</tr>
<tr>
<td>Density</td>
</tr>
</tbody>
</table>
5.1 Conclusions

To conclude, we have demonstrated the necessity for specialized search algorithms for tuning the hyper-parameters of a machine learning model. We studied two search algorithms, genetic algorithm and particle swarm optimization, and have compared their performance with the traditional methods of tuning hyper-parameters, namely, grid search and random search. We have shown that genetic algorithm and particle swarm optimization clearly outperform grid search and random search by a huge margin, based on the mean absolute error for prediction. Both feature selection via genetic algorithm and a reduced data set for training are able to reduce the computational time of hyper-parameter optimization without compromising much on the prediction accuracy. We also implemented these two methods in the hyper-parameter tuning module in the ChemML suite which can be used for tuning the hyper-parameters of any given machine learning model.
5.2 Challenges to Hyper-parameter Optimization

Although the machine learning models tuned by both genetic algorithm and particle swarm optimization have better prediction accuracy than grid search and random search, they fall short on the time aspect of these computations (genetic algorithm: 1.5 days, particle swarm optimization: 4.3 days). Furthermore, these results are more often than not dependent on the data sets given to the model. A search algorithm performing better on one data set might not be ideal for optimizing a different data set. There is no ‘one-algorithm-fits-all’ solution for hyper-parameter tuning.
Bibliography


