2022

# A SURROGATE MODEL OF MOLECULAR DYNAMICS SIMULATIONS FOR POLAR FLUIDS: SUPERVISED LEARNING METHODS FOR MOLECULAR POLARIZATION AND UNSUPERVISED METHODS FOR PHASE CLASSIFICATION

Zackerie W. Hjorth
*Michigan Technological University*, zwhjorth@mtu.edu

# A SURROGATE MODEL OF MOLECULAR DYNAMICS SIMULATIONS FOR POLAR FLUIDS: SUPERVISED LEARNING METHODS FOR MOLECULAR POLARIZATION AND UNSUPERVISED METHODS FOR PHASE CLASSIFICATION

By

Zackerie W. Hjorth

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Physics

MICHIGAN TECHNOLOGICAL UNIVERSITY

2022

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Physics.

Department of Physics

| | |
|---|---|
| Report Advisor: | *Dr. Issei Nakamura* |
| Committee Member: | *Dr. Ranjit Pati* |
| Committee Member: | *Dr. Elena Giusarma* |
| Department Chair: | *Dr. Ravindra Pandey* |

# Contents

**List of Figures** . . . . . **vii**

**Abstract** . . . . . **ix**

**1 Introduction** . . . . . **1**

1.1 Problem Domain . . . . . 1

1.2 Data Collection . . . . . 2

**2 Deep Neural Network Ensembles** . . . . . **5**

2.1 Method . . . . . 5

2.2 Results and Discussion . . . . . 8

**3 Support Vector Machine Regression** . . . . . **11**

3.1 Method . . . . . 11

3.2 Results and Discussion . . . . . 13

**4 Unsupervised Clustering for Phase Classification** . . . . . **15**

4.1 Method . . . . . 15

4.2 Results and Discussion . . . . . 17

**5 Conclusions** . . . . . **23**

**References** . . . . . **25**

# List of Figures

2.1 DNN Architecture . . . . . 7

2.2 DNN Training History . . . . . 8

2.3 DNN Heatmap of Dielectric Constant . . . . . 9

3.1 SVR Heatmap of Dielectric Constant . . . . . 13

4.1 Agglomerative Dendrogram . . . . . 17

4.2 K-Means Clusters . . . . . 18

4.3 Phase Boundary for DNN and SVR . . . . . 18

4.4 Simulation Snapshots . . . . . 19

4.5 Four Clusters for SVR . . . . . 21

# Abstract

Molecular Dynamic (MD) simulation is a standard computational tool in soft matter physics. While very powerful, it is computationally expensive, leading to some simulations taking days or even weeks to complete depending on the size of your computer cluster. Finding computationally cheap surrogate models which can learn the output features of MD simulation is therefore highly motivated. In this report I explore the use of deep neural network ensembles as well as support vector machine regressors as surrogate models for MD simulation. From the output of the surrogate models, we can then employ unsupervised learning methods to get insight into the physics of our system, and classify boundaries between phases. We will also show the potential of this method to uncover behavior not realized by other methods.

# Introduction

## 1.1 Problem Domain

Molecular Dynamics (MD) simulation is a ubiquitous computational tool in the field of soft matter physics research as one of the main ways that scientists study the properties of soft materials. While MD simulation has incredible predictive power, it comes at the cost of long computation times. Running a single simulation can take anywhere from a few hours, to a few days depending on the parameters of your simulation and the size of the cluster you run it on. When many simulations need to be done in order to complete a study, it is clear that finding ways to either boost the speed of computation or reduce the number of simulations needed to be performed is highly motivated. In this report, I will be exploring multiple machine learning methods to predict the output of a simulation given the input parameters.

This will allow researchers the ability to use these trained machine learning models as surrogates for the MD simulation. I will also show that by employing machine learning to describe the output of the simulations, one can get meaningful insights into the system that might have otherwise been missed.

In this report, I will be focusing on simulations of Stockmayer fluids. Stockmayer fluids are fluids whose constituent particles are approximated as spheres with a permanent dipole. While this model grants computational simplicity, I now have two parameters that I need to fix for the simulation. There is no perfect way to reduce the complex structure of a molecule to the simple model of a sphere with a permanent dipole. This means that many simulations need to be performed in order to fix these parameters. With my machine learning methods, I will be attempting to predict the square of the polarization of the system given the diameter and dipole moment of the molecule in question.

## 1.2 Data Collection

For this research project, we collected data from 33 MD simulations of a liquid with varying diameter and dipole moment. The values for dipole moment ranged from 1.6-2.3 debye, and the diameter ranged from 2-3 angstroms. The box size of these simulations was $9261 Å^3$ and each simulation contained 310 molecules. The output

variable I will be trying to predict is the square of the polarization. These simulations were done with low accuracy, and very short run times (2000 samples). This means that the results of each simulation are unlikely to have a good statistical convergence. The accuracy of each simulation is not the goal for this project. I am trying to see if machine learning can accurately learn the output of our simulations, whether they are accurate or not.

# Deep Neural Network Ensembles

## 2.1 Method

Our first attempt at using machine learning to predict the square of the polarization of the simulations utilized a deep neural network (DNN). DNNs are known for their power and flexibility, as they are arbitrary function approximators that can be used for regression or classification [1]. For this first attempt, I wanted to assess how well a machine learning framework could perform if simply given raw output files. As stated previously, I have 33 simulation files with 2000 samples each. For this first attempt, I decided not to scale the data. An issue I encountered from the beginning was that DNNs take a high volume of data in order to be trained. Rather than averaging the data for each of the 33 simulations, I instead chose to treat each snapshot as a unique data point. I believed that this would allow our DNN to train

well while learning to approximate the mean of the square of the polarization for each simulation anyway. The hyperparameters of the DNN were selected by trial and error. The DNN I created has four hidden layers, with 64 neurons in each of the first three hidden layers, and 32 neurons in the last hidden layer. As our network is somewhat deep, I used batch normalization between the layers to ensure that training went smoothly [2]. ReLu functions were used as the activation functions of the hidden layers, while the activation of the output neuron was linear. In order to set the number of epochs, I simply printed the training and validation history and used the elbow method. That is, I visually inspected where the improvement of the performance of the network rapidly decreased, and checked to see if the training and validation accuracy diverged from each other. Then I set the number of epochs equal to this point so as to prevent over-fitting. Through this trial and error I found that using mean squared error (MSE) as the cost function performed well for data from the regions with a high polarization, but that it performed poorly for the disordered regions. Conversely, using mean absolute percentage error (MAPE) worked well in the disordered regions but not in the highly polarized regions. I then decided to create an ensemble of DNNs, one using MSE and one using MAPE. A combiner neural network took in both the diameter and dipole information as well as the output from both of the constituent networks. Ensemble methods have been proven to be better than using a single model, and benefit from the constituent models being diverse [3]. The diversity in our ensemble comes from difference in loss function. I have experimented

with methods of bagging, meaning each constituent network is given a random subset of the training data, however those results are not reported here. The structure of the combiner network is shown in Figure 2.1.

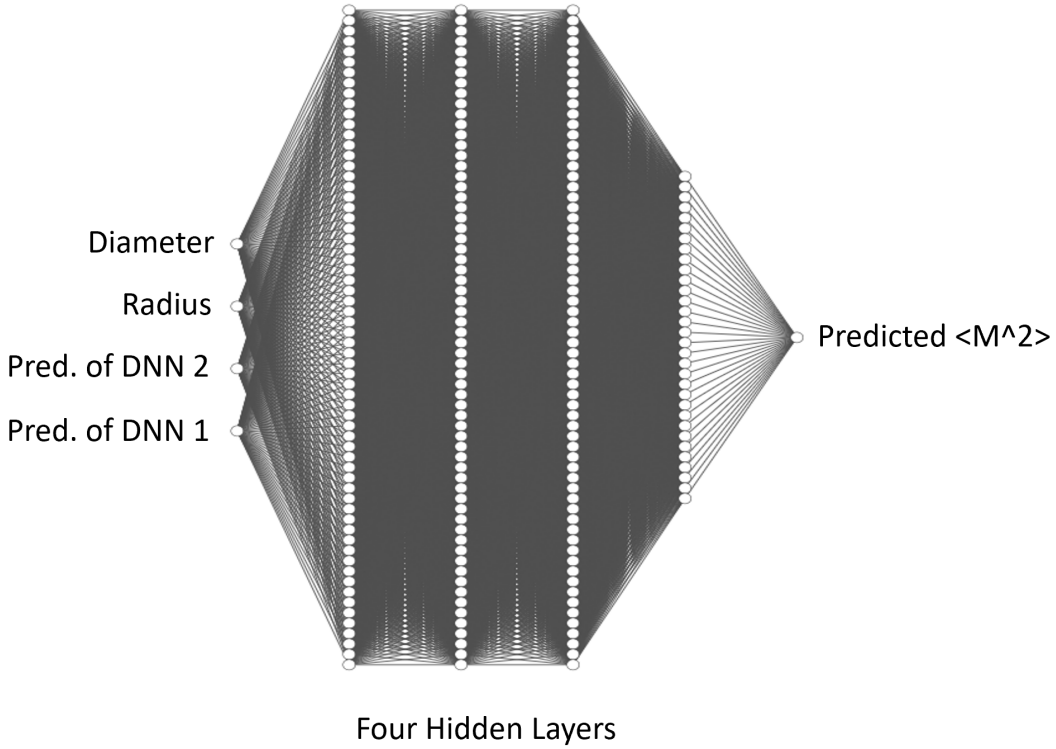


**Figure 2.1:** Shown is the structure of the combiner network used in the DNN ensemble.

The output of the combiner network was then the prediction I used. I also used the elbow method to determine the number of epochs to train the combiner network just as I did for the constituent networks. As you can see on the history plot in Figure 2.2, the training of the combiner network is now much more stable than any of the constituent networks individually.

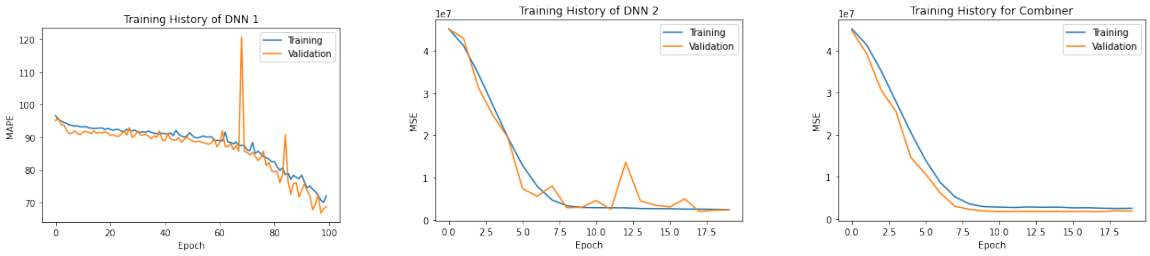


**Figure 2.2:** Pictured are the training histories of each DNN in the ensemble. Note that the combiner network is not prone to large jumps in the loss function like the constituent networks are.

## 2.2 Results and Discussion

The resultant plot of the predicted dielectric constant (calculated from the squared polarization values) in the parameter space is shown here in Figure 2.3. We also see a stark division line between a region of very high polarization and a region with a very low polarization. I believe this might be indicative of a phase change. This will be discussed in a later section.

This method seems to work well, however, there are several concerns. The largest issue is that by treating each point in the simulation data as unique, rather than averaging the values, we have muddied up the test and train sets. The golden rule of machine learning is never look at the train set while testing. However, each of our points have information about the test points as they came from the same simulation. This is evidenced by the fact that we do not see a divergence of the test and train accuracy after the elbow in our training history plots. In addition to this, our network

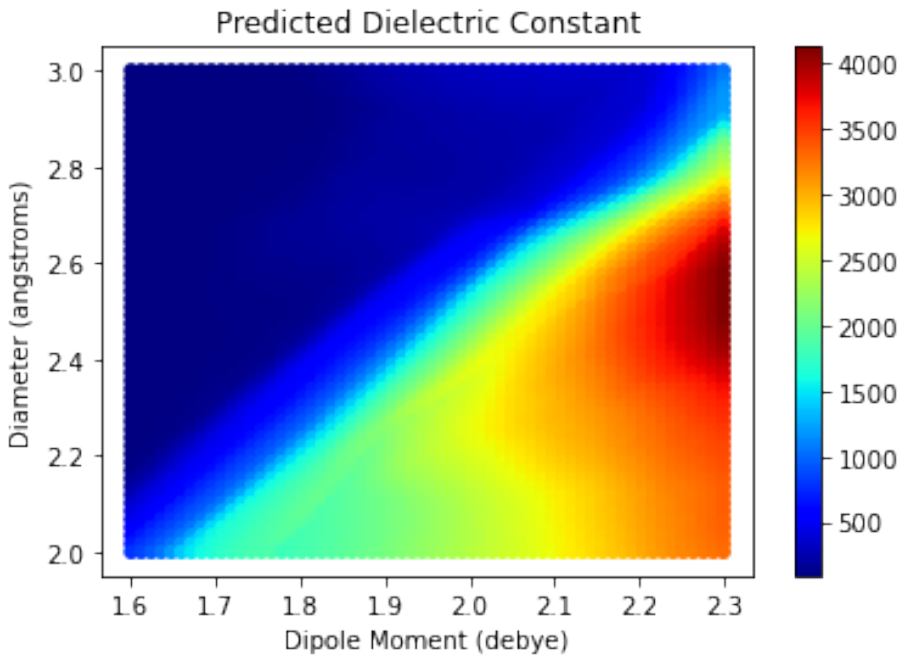


**Figure 2.3:** Heatmap of predicted dielectric constant values over the parameter space using a DNN ensemble. Dielectric constant was calculated using the mean of squared polarization.

needs to be quite large in order to make good predictions. I believe this is because the value of our target variable is so extreme. Finally, I did not scale the input data. This was done because our early motivation was to make this method as simple as possible. However, it is always standard in machine learning to scale our input features. This may not be necessary in our case though, as our variables are on roughly the same order as each other, therefore the importance of a feature wont be arbitrarily inflated. I did experiment with scaling the input features early on and saw no real measurable benefit, therefore I am inclined to believe it is not necessary in this case. These issues will be addressed in the next method.

# Support Vector Machine Regression

## 3.1 Method

For the next attempt at using machine learning to predict the output of MD simulation, I decided to average the data from each simulation giving us a total of 33 data points. In order to make the most of this data, I decided to use a support vector machine regressor (SVR). I chose this framework as kernel methods are a classic choice for problems with a small data set. A conventional support vector machine (SVM) is a classification method that attempts to separate different classes of points by a hyperplane, and maximizing the margin between the hyperplane and the nearest points. Obviously, almost all classification problems we encounter are not linearly separable. We therefore employ what is known as the kernel trick, and use some kernel (linear, polynomial, gaussian, etc.) to transform our data into a higher dimension where the

points will be linearly separable. I can use a very similar framework to this for regression as well, SVR [4]. The difference with SVR is that we are trying to maximize the number of points within the margin of the hyperplane, and use this hyperplane as the prediction of the model. For this approach, I decided to scale both the input features as well as the target variable. Fitting the transformer was done only on the training data so as not to leak global features of the data set to the training set. For the target variables, I used the standard scaler from scikit learn, which gives the data a mean of zero and unit variance. As for scaling the input features, I decided to use a pipeline so that I could treat the scaler as a learnable parameter. In order to find the best parameters, I used a grid search with k-fold cross validation. I was searching through three scalers, (standard, minmax, and robust), four kernels (polynomial, rbf, sigmoid, and linear), and various parameters for each of these kernels. To determine the best set of parameters, I used the mean MSE across the folds. I found that the best model used a gaussian kernel, a regularization parameter (C) of 1.2, and used the standard scaler. It should be noted that the particular scaler used did not seem very important, as the predicted best scaler would often vary between runs. I then retrained the network using these best parameters.

## 3.2 Results and Discussion

Using these parameters, I get a heatmap of the dielectric constant shown in Figure 3.1. The test accuracy of this model was 86%.

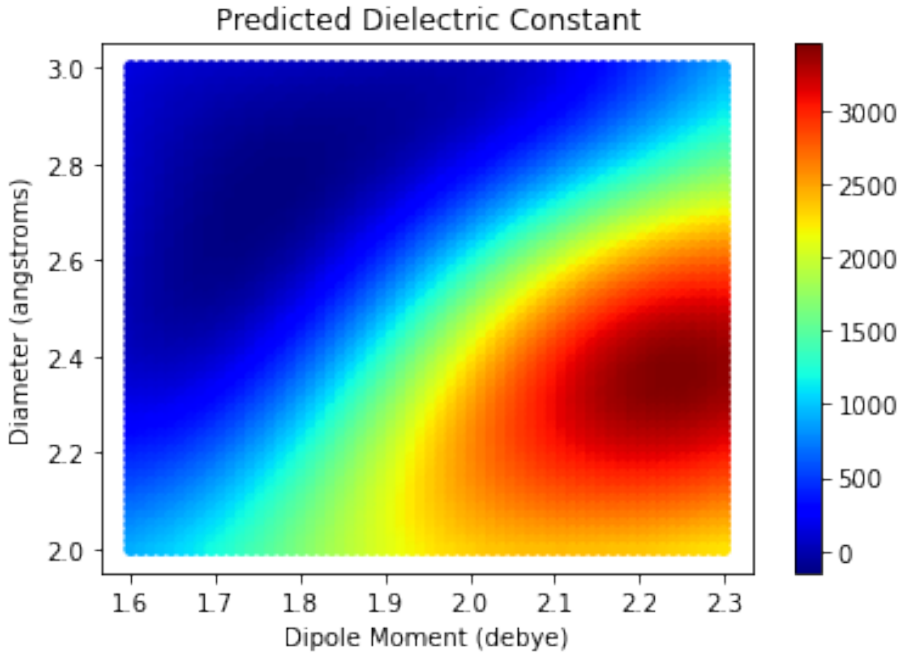


**Figure 3.1:** Heatmap of predicted dielectric constant values over the parameter space using SVR. Dielectric constant was calculated using the mean of squared polarization.

As can be seen, this shows very similar behavior to the DNN method, though the region with very high polarization is a bit different. This may mean that our DNN method was sound. Once again, there seems to be a phase transition.

# Unsupervised Clustering for Phase Classification

## 4.1 Method

Now I will look at the phase behavior of our predicted outputs. I will use clustering techniques in order to achieve this. Clustering is an unsupervised machine learning method where we try to classify data into k classes without having labels to check. I will be using two methods of clustering, which each give similar results. The first and most common method is called k-means clustering. In k-means clustering, the number of classes I will be splitting the data into is a hyperparameter. The algorithm works by selecting k randomized centroids, and classifies each data point as belonging the class of the centroid it is nearest to (using whichever metric you specify). Once that

is done, move the centroids to the center of their cluster. I then reclassify the points, and repeat. This method aims to minimize the in-cluster variance, and maximize the variance between clusters [5].

The other clustering technique I will use is hierarchical clustering, specifically agglomerative clustering. Hierarchical clustering assumes that there is a hierarchy of classes, and either works from the bottom up to merge points into classes one by one until there is only one class remaining (agglomerative), or top down assuming all points belong to the same class and splitting the classes repeatedly until all points belong to their own class (divisive). The hierarchy of classes can then be seen by reporting the dendrogram of the linkages. This method is not as fast, nor as common as k-means clustering, but it allows us to not have to treat the number of classes as a hyperparameter. We can look at the dendrogram to visually see the evidence for how many classes make up the data. I then trim the dendrogram and split the data into that many classes.

For my research, I saw that there was very little difference between the hierarchical clustering and the k-means clustering, so I simply used the created dendrogram to justify the number of phases I chose to fit. This is done by finding the largest vertical distance on the dendrogram that does not have an instance of classes being merged. This only guarantees the best parameters if our data satisfies the ultrametric triangle inequality, which I have no reason to believe is the case for our data [6]. I therefore

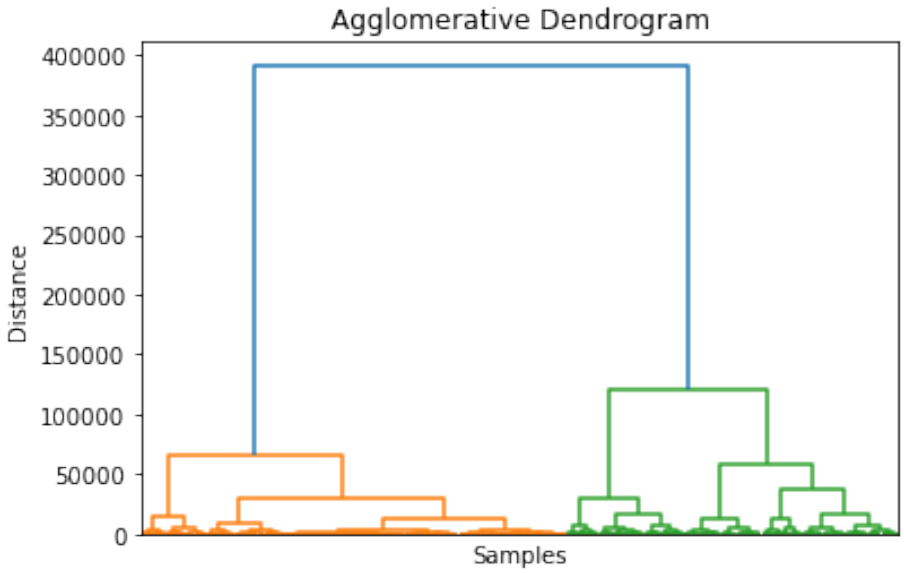


**Figure 4.1:** The vertical axis of the dendrogram represents the "distance" between clusters. The vertical distance one has to go before clusters are merged indicates the difference between these clusters.

may choose to trim the dendrogram in a different fashion if I have good reason to do so. The dendrogram produced is shown in Figure 4.1. As can be seen, there is strong evidence to support the existence of two phases.

## 4.2 Results and Discussion

Here are I report the results of each of the methods. As we can see in Figure 4.2, the clustering methods picked up on the same features as we did visually when looking at the heatmaps. By finding points on the boundary between classes, I can use this to generate a predicted phase boundary, shown in Figure 4.3. In order to see if this is a reasonable division, I fit a series of sigmoid curves to the data and used

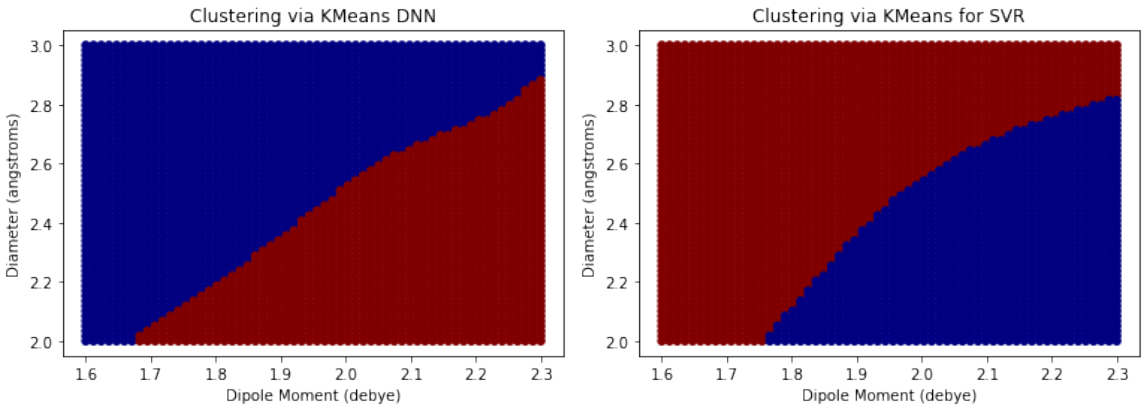


**Figure 4.2:** Clusters generated on data from both surrogate models seem very similar (ignore the inverted colors, color choice was arbitrary).

their midpoints as the boundary between phases. As you can see, there is quite good agreement between the two methods. I believe that means this method works well, and will be much easier to generalize for multiphase diagrams than the sigmoid method.

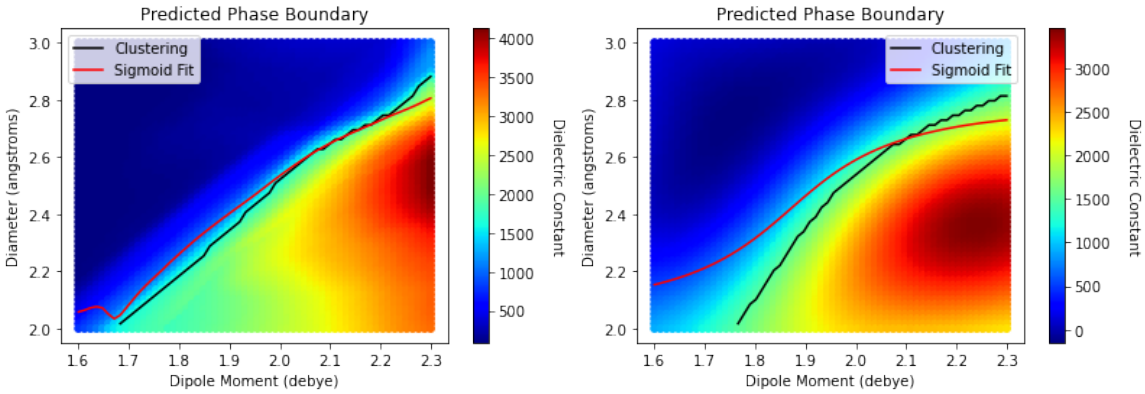


**Figure 4.3:** For the DNN surrogate model, there is little difference between the boundaries calculated from fitting sigmoids and via clustering. The difference between the methods is more apparent for the SVR surrogate model.

One potential issue with this method is that I did not scale the input features. As said previously, this is because they were roughly on the same order as each other. It

must be noted, however, that this means these clusters are not invariant to the units used to measure the input features, and this could delay computation time for larger data sets [7]. I do not expect this to be much of an issue, for the reasons previously stated.

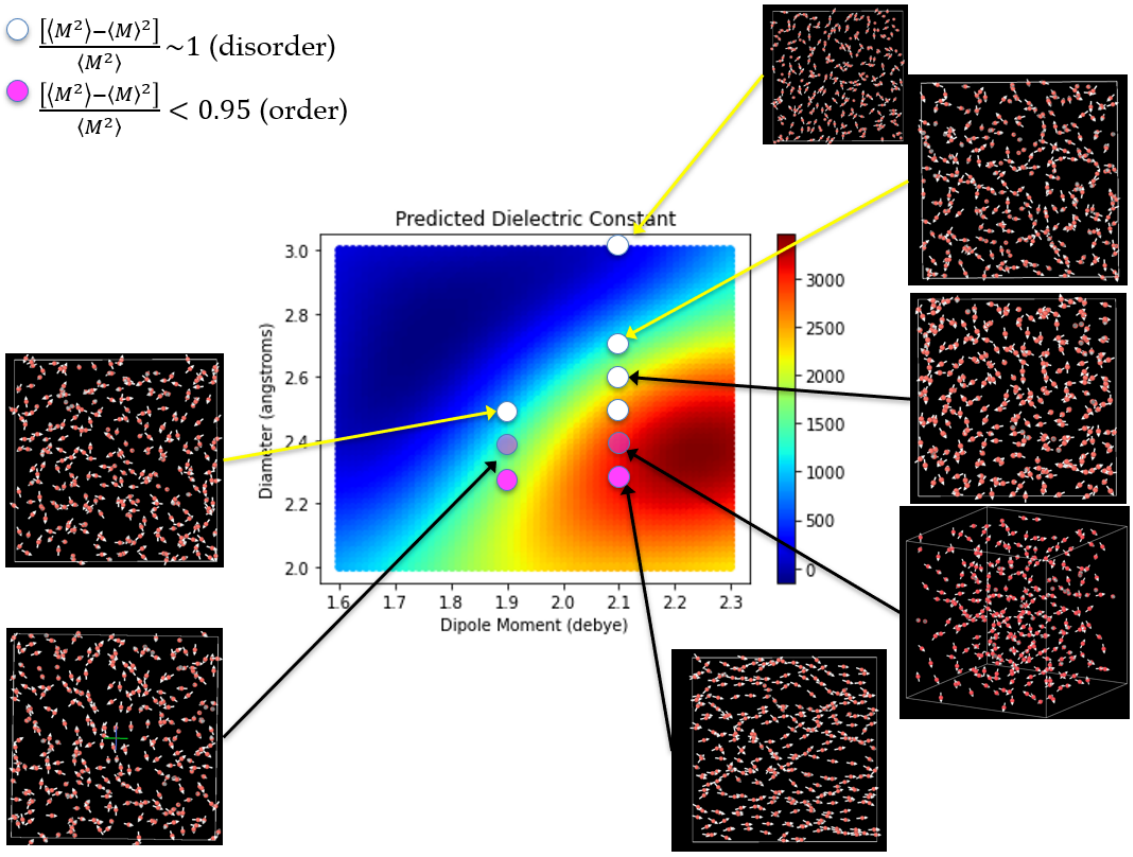


**Figure 4.4:** Shown are some MD simulation snapshots taken from various regions in the feature space. A dark pink dot represents a highly ordered phase, and we can see in some snapshots, the dipoles are clearly aligned.

Now that I have a predicted phase boundary, I can compare the results to some simulation snapshots as shown in Figure 4.4. Here, a low ($< 0.95$) order parameter means that I am predicting the existence of an ordered phase. As there is a divergence in the dielectric constant, I believe this to be the ferroelectric phase transition [8]. We

can see the regions with a low order parameter are in fact contained within the region where I predict a highly polarized phase. However, regions with among the highest polarization are predicted to be in a disordered phase. What is happening here? One potential explanation is that in the regions with a very high predicted polarization, the dipoles may be linking up to form loop structures whereas the regions in the bottom of the graph are forming long straight chains. This would mean that in the red region there may exist local zones of high polarization, but with a global average near zero. It is known that dipoles in Stockmayer fluids can make these loop structures [9].

When looking at the simulation snapshots, it is difficult to tell whether or not any loop structures are forming. In order to test this, I will need to calculate the vorticity of the simulation box, which I have not yet done. If this is the case, it may be more accurate to say that there are four phases existing in this parameter space: a disordered phase, a fluctuating (transition) phase, a globally ordered phase (long chains), and a locally ordered phase (loops). I can then fit four clusters to our data and see if this matches up with our expectations. I then used k-means clustering to group the data into four clusters as shown in Figure 4.6. If I am correct and the region with a high predicted dielectric constant corresponds to a region of high local polarization, then I have successfully shown that this machine learning technique can be used to find the existence of phases that were not previously being classified.

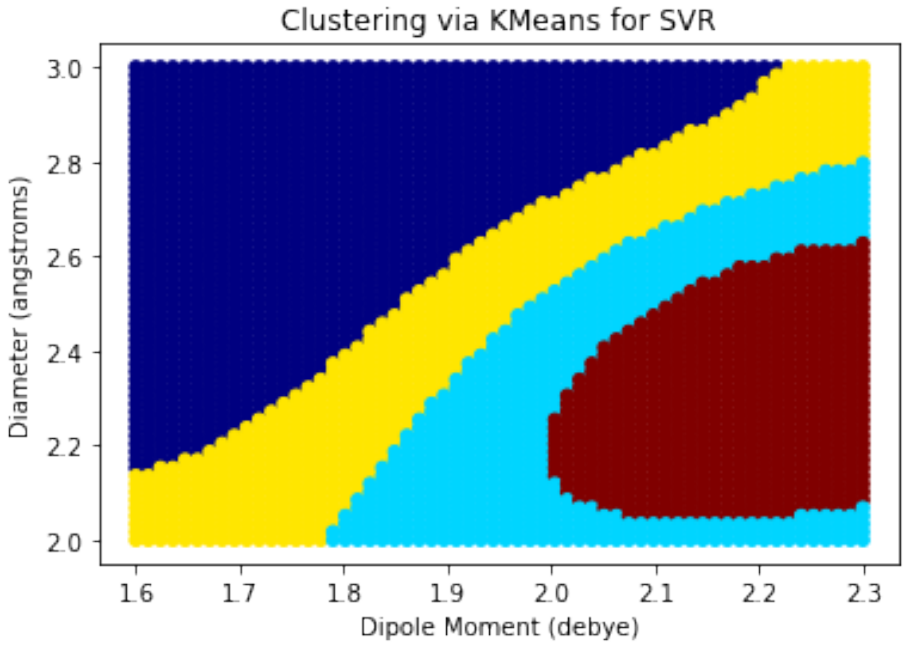


**Figure 4.5:** Phase diagram generated using four clusters. From left to right, these phases may represent a disordered phase, a fluctuating phase, a globally ordered phase, and a locally ordered phase.

# Conclusions

In conclusion, I believe that I have shown machine learning to be capable of learning the output of MD simulation given the input parameters. Both DNN ensembles and SVRs seem like viable candidates for generating surrogate models of MD simulation. Overall, I expect that the performance of the SVR model to be more reliable as it was trained on the averaged data from simulations and did not have an issue with data leakage. Using the outputs of these surrogate models, I can use clustering techniques in order to determine the existence of phases in our system and to classify the boundary between those phases. I found strong evidence of a disordered and ordered phase indicating the ferroelectric phase transition. Within the region where I predicted a large polarization, we can see potential evidence that there is the existence of a phase exhibiting strong local polarization, but not global polarization. If this is the case, this is a feature that was missed using traditional techniques alone. This suggests that machine learning is an important tool for analyzing these soft matter

systems.

# References


[1] Gühring, I.; Raslan, M.; Kutyniok, G. *CoRR* **2020**, *abs/2007.04759*.

[2] Batch normalization: Accelerating deep network training by reducing internal covariate shift. Ioffe, S.; Szegedy, C. **2015**.

[3] Granitto, P.; Verdes, P.; Ceccatto, H. *Artificial Intelligence* **2005**, *163*(2), 139–162.

[4] Brereton, R. G.; Lloyd, G. R. *Analyst* **2010**, *135*, 230–267.

[5] Ahmed, M.; Seraj, R.; Islam, S. M. S. *Electronics* **2020**, *9*(8).

[6] Murtagh, F.; Contreras, P. 01 **2012**, *2*, 86–97.

[7] Mohamad, I. B.; Usman, D. *Research Journal of Applied Sciences, Engineering and Technology* **2013**, *6*(17), 3299–3303.

[8] Bartke, J.; Hentschke, R. *Molecular Physics* **2006**, *104*(19), 3057–3068.

[9] Computer simulation of the stockmayer fluid. Bartke, J. **2008**.