Predicting the Infinite Dilution Activity Coefficient with Machine Learning

Guido Petri

Technical University of Berlin
Institute of Process Engineering

June 4th, 2019

Advisors
Prof. Dr.-Ing. Jadran Vrabec - Technical University of Berlin - vrabec@tu-berlin.de
Dr.-Ing. Erik Esche - Technical University of Berlin - erik.esche@tu-berlin.de
Dr. Ian Bell - National Institute of Standards and Technology - ian.bell@nist.gov
Dr.-Ing. Andreas Jäger - Technical University of Dresden - andreas.jaeger@tu-dresden.de

Abstract

The infinite dilution activity coefficient plays an important role in the separation of mixtures. In order to improve our understanding of the thermodynamic properties of mixtures and focus our research on the molecular species most likely to yield good results, we explored several different machine learning estimators and their performance when predicting the slope of the infinite dilution activity coefficient at approximately room temperature for 1735 molecular species. A comparison was made between the estimators' performance with and without COSMO-SAC electronic patch data. The estimators' performance with and without a principal component analysis was also examined. Goodness-of-fit scoring distributions show that the patch data provides meaningful improvements to all of the estimators. However, while the electronic patch data improves the goodness-of-fit score, the estimators performed at nearly the same level with exclusively atomic data. The most accurate estimators were the non-linear models. This implies that the IDAC is not a linear combination of the chosen features, but functions more akin to a decision tree.

Table of Contents

Introduction

In the past few decades, much progress has been made towards automated pattern-matching with computers. The umbrella term of “machine learning”[26] describes the methods underlying this field of science: a machine is taught what its input looks like and provides an output based on these. The two main areas of supervised and unsupervised machine learning divide these methods into a

group where the output is known in advance, the machine being taught to predict this output, and a group where the output is learned organically by the machine. In supervised learning, there are once again two main areas: regression and classification. Regression describes a continuous-valued output, whereas classification describes a discrete, binned output that is not necessarily numerical.

These forms of machine learning have been applied to numerous different fields, including image recognition[27], stock predictions[28] and biomedical research[29]. In this work, we aim to re-introduce machine learning into the field of thermodynamics and chemistry by comparing the performance of different regression estimators on a COSMO-SAC[38][39] dataset and on an exclusively atomic data dataset, with the output for both datasets being the *Infinite Dilution Activity Coefficient* (IDAC)[30][31][32] slope.

The IDAC is of value due to the ability to extrapolate interaction energies from it; in particular, its use in the development of separation processes is of great importance to the chemical industry[30], since one can derive the Henry's law coefficient[31][32] and thus P-T-x-y data from the IDAC. The IDAC has long been of importance to the chemical world[33], and thus, it makes sense to explore possibilities of predicting the IDAC without requiring experimental data.

Several models for activity coefficient prediction already exist, some of which are UNIFAC[34], based on a molecular species' functional groups, and COSMO-SAC, based on a molecule's electronic field. Both of these have versions that improved through further research and fine-tuning[35][36], but of course have advantages and disadvantages; for one, they are computationally expensive, and do not provide perfect accuracy[37]. Some processes from the field of machine learning could be used to add another activity coefficient predictor to the chemist's toolbox. In particular, it would be of great use if machine learning could be used to predict the IDAC starting from purely atomic data, that is, well-known and accurate thermodynamic data that is generalizable across molecular species and across functional groups.

In this thesis, we received COSMO-SAC data and atomic data for over 2233 molecular species, mostly composed of carbon, hydrogen and oxygen. We also received IDAC data for the temperature of 300 K and 302 K. As such, this work has limited application: the findings cannot be extrapolated to non-organic molecular species, nor temperatures far from 300 K. The objective of this thesis is to compare the performance of the machine learning estimators on exclusively atomic data with that of the estimators on COSMO-SAC patch data, in order to establish whether it would be viable to forgo the COSMO-SAC data for an extrapolation of the IDAC. This would result in a predictor that is not particularly expensive computationally, but would provide a rough estimate with sufficient accuracy to filter out molecular species that are not a good fit for a particular purpose, whether this be a low IDAC slope or a high IDAC slope.

The methodology for creating our datasets is explained, followed by a short description of each regression method we employed. The standard methodology for machine learning exercises is explained. Then, the results between regressors for a dataset and between datasets themselves are discussed. Finally, we elaborate on what possible improvements can be made, and in which direction research in this area should focus.

Methods

Dataset Creation

We received the COSMO-SAC data for 2233 different molecules. This data was composed of three files: the first containing information on electronic patches, the second containing information on the atoms, and the third containing a predicted IDAC, at the temperatures of 300 and 302 K, based on

the COSMO-SAC data. The patch data included area of the patch, its charge, which atom the patch belongs to, and the patch radius, as well as its location in Cartesian coordinates. The atomic data was composed of the element the atom belongs to, the atom indices, and its location in Cartesian coordinates. We appended the atomic mass, the Pauling electronegativity, the atom type - metalloid, metal, non-metal -, the stable isotope count, the incidence of the most stable isotope in percentage, the total isotope count, the electron count, the empirical radius, the neutron-to-proton ratio, the maximal atomic charge, the minimal atomic charge, and the first ionisation energy of the atom.

Out of this data, we fashioned an origin dataset for our experiment. The features engineered in it are presented in the following table:

Table 1: The features engineered for each molecular species, by data source. The machine learning estimators used these in order to predict the IDAC slope.

Atom-sourced features	**Patch-sourced features**
total atom count	minimal/average/maximal patch charge
count of carbon/hydrogen/oxygen/other atoms	total charge of a molecule
minimal/maximal Manhattan distance between atoms	total patch surface area
minimal/maximal/average electronegativity of the atoms	patch count
most common occurring element in the molecular species	largest charge difference between patches (dipole moment)
number of stable isotope variations	center of charge in Cartesian coordinates
incidence of the most common isotope combination in percentage	
total number of isotope variations	
minimal/maximal/average proton count	
minimal/maximal/average atomic radius	
average neutron-to-proton ratio	
maximal/minimal atomic charge possible	
minimal/maximal/average ionisation energy	
total mass	
molecular volume	

For 825 of the molecular species, the molecular volume was not available. The volume for these molecular species was estimated using a *k*-Nearest Neighbors algorithm, with the five closest neighbors, the Euclidean distance, and uniform weights. A flag was added to the data to identify molecular species for which the volume was estimated in this fashion.

In the preprocessing stage, we removed outliers using the 1.5 interquartile range (IQR) rule. This left us with 1735 molecular species. Finally, in order to reduce the possible effect of large ranges

and skew for any given dimension, the dataset was normalized with a Yeo-Johnson power transformation[1]

$$\psi(\lambda, y) = \begin{cases} \frac{((y+1)^{\lambda}-1)}{\lambda} & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y+1) & \text{if } \lambda = 0, y \geq 0 \\ \frac{-((-y+1)^{(2-\lambda)}-1)}{2-\lambda} & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y+1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

where λ is the power parameter, bounded by 0 and 2, y is the value to normalize, and ψ is the normalized value, a function of λ and y. Our values for λ were chosen independently for each feature and automatically, in a manner such as to maximize the profile log likelihood.

Four datasets were then created from this root dataset. The first, named *all*, includes all the features engineered from the original COSMO-SAC data, as well as features derived from atomic data. A second dataset, named *atom*, includes only the features that can be derived exclusively from atomic data, that is, without any information on the electron probability patches available through COSMO-SAC. The third and fourth datasets, with the suffix *_pca*, are the first two datasets having undergone a *principal component analysis* (PCA)[41]. This process finds a linear combination of the dimensions present in the data that best describes its variability, in order to reduce the dimensionality of the data. In the creation of the PCA datasets, we removed dimensions until only 75% of the variability was still explained by the new, linearly combined dimensions.

We examined the performance of ten estimators[13] in our experiment.

Ridge Regression

The ridge regression[2] is similar to an ordinary least-squares (OLS) linear regression. In some cases of an ordinary least-squares regression, multiple solutions exist; these are called ill-posed problems, since they do not conform to Jacques Hadamard's definition of a well-posed problem[40]. In order to counteract this and provide a unique solution, the problem must be re-formulated so that we have a stable algorithm to solve. Ridge regressions do this via a regularization term which gives preference to more particular solutions. Where OLS regressions minimize the sum of the squared residuals

$$\|y - Ax\|_2^2$$

the ridge regression minimizes the following

$$\|y - Ax\|_2^2 + \|\Gamma x\|_2^2$$

with y as the vector of target values, x as the vector of coefficients, A as the input matrix and $||x||_2$ indicating the Euclidean norm. The regularization term includes the factor Γ, which is called the *Tikhonov matrix*[2]. This is frequently chosen as a multiple of the identity matrix. In our solutions, the Tikhonov matrix is directly derived from the hyperparameter α

$$\Gamma = \alpha I$$

where I represents the identity matrix and α is an integer value. This regularization method is also known as L2 regularization. For $\alpha = 0$, the ridge regression reverts to a OLS regression.

Random Forest Regression

Random forests are an ensemble method[21][22][23], meaning they are composed of several sub-estimators. Each individual estimator is a decision tree which takes different branches depending on the values for particular dimensions. The decisions themselves are chosen by the computer in order to maximize variance between its two branches. However, since a single decision tree is prone to overfit to the data - that is, it is prone to creating branches until only a single data point is found at each "leaf" -, several of these decision trees are created based on samples taken from the training data and their estimate for the test value is averaged. This helps prevent that overfit, i.e., that the estimator learns exclusively its train data and isn't generalizable to a test dataset. Since the trees are not correlated, they can be grown in parallel, which is a computational advantage.

Available hyperparameters for managing the estimator include the number of estimators n, the maximum depth of a tree, the minimum number of samples at a leaf n_{Leaf}, the loss criterion to use between mean squared error and mean absolute error, and the maximum number of dimensions to look at when looking for the best split. The trees can also be built upon the entire dataset instead of upon a sample of the training data, but this causes the trees to be very correlated, increasing the bias of the estimator.

Lasso Regression

The lasso regression[4] is also similar to the OLS linear regression. The difference is once again a regularization term. Instead of minimizing the sum of the squared residuals

$$||\mathrm{y}-\mathrm{Ax}||_2^2$$

a Lasso regression minimizes the following

$$\frac{1}{2\cdot \mathrm{n}}||\mathrm{y}-\mathrm{Ax}||_2^2+\alpha||\mathrm{x}||_1$$

with n being the number of samples in the dataset, and $||x||_1$ representing the absolute value norm. This is similar in format to the ridge regression, but instead of a squared regularization term, we minimize the sum of an absolute value. This regularization format is named L1 regularization[3][4][6]. L1 regularization also reduces the effective number of features, since it yields many coefficients equal to zero. This is also directly controlled by the selection of the parameter α, which is once again a hyperparameter in our solution. Similarly to the ridge regression, the solution reverts to an OLS regression with $\alpha = 0$.

ElasticNet

The ElasticNet regression[5] is a combination of the lasso and the ridge regressions. It uses both L1 and L2 regularization in order to solve the linear model[5][6][7]

$$\frac{1}{2 \cdot n} \| y - Ax \|_2^2 + \alpha \, L_1 \, \| x \|_1 + 0.5 \alpha \, (1 - L_1) \, \| \Gamma x \|_2^2$$

where L_1, bounded by 0 and 1, represents the ratio of regularization as per L1 or L2. Together with α, this is also a hyperparameter which we provide to the model. For $\alpha = 0$, this also collapses into a OLS regression, and for $L_1 = 1$ or $L_1 = 0$ it collapses to a lasso and ridge regression, respectively.

Least Angle Regression

The least-angle regression, also known as LARS[8][9], operates similarly to a forward stepwise regression. Every dimension's coefficient starts at zero. At each step of the regression, it finds the dimension that is most correlated with the target and increases its coefficient. If there is more than one dimension with the most correlation, an equal angle between the two features is chosen instead. The method is usually continued until all the dimensions are involved in the estimator. In our case, we can control the number of nonzero coefficients via a hyperparameter. In this method, correlated dimensions have correlated growth of their coefficients. This represents a problem for high-dimensional data, since it is probable that there are dimensions with collinearity, even to a small degree. The LARS method is also very sensitive to noise, since it is based on refitting upon the data iteratively.

k-Nearest Neighbors Regression

The *k*-nearest-neighbors regression[18] is centrally based on a distance calculation. For every data point, the distance to all other data points is calculated and the *k* nearest neighbors are selected. The output is then either a simple or weighted average of the neighbors' results. Since calculating the distance for *n* data points involves *n!* calculations, shortcut methods have been developed that provide faster processing at the cost of accuracy. For example, if two points A and B are very distant from each other, and a third point C is near point A, then it isn't necessary to calculate the distance between points B and C, since logically they will also be very distant. In our experiment, the two shortcut methods we had available were *k*-dimensional trees and ball trees.

The *k*-dimensional tree bases itself on a similar idea to two-dimensional quad-trees, or three-dimensional octrees. The dimensional space is recursively divided into subregions, with greater detail being given where there are more data points[19]. This subregion construction occurs quickly, and once it is done, the distance between two data points can be calculated much quicker. However, this method is limited in applicability due to it becoming inefficient when there are more than 20 dimensions in the data.

The ball tree addresses these high-dimensional inefficiencies. Where the *k*-dimensional tree divides the data into subregions along the dimension axes, ball trees divide the data into hyperspheres. The tree construction takes longer, but the resulting structure provides the advantages of *k*-dimensional trees at a high dimensionality[20].

Both of these methods are also able to revert to brute-force distance processing via a hyperparameter named *leaf_size*. This is useful, since the tree calculation can be more expensive than the brute-force approach when there are several data points clustered together.

The distance between two points can also be calculated in several ways, the most common of which are the Manhattan distance and the Euclidean distance. In our experiment, we had a hyperparameter *p* available for the Minkowski distance, which with $p = 1$ is equivalent to the Manhattan distance, and with $p = 2$ is equivalent to the Euclidean distance.

Gradient Boosting Regression

Gradient boosting[24], similar to the random forest, is also based on decision trees, but unlike its counterpart, these trees are shallow and sequential. The estimator reduces bias - the difference between the average prediction of the model and the target itself - as much as possible. It accomplishes this by progressively adding decision trees, fitting against the negative gradient of a loss function[25].

The decision trees used for estimating are termed *weak learners*, since they are very variable and individually, do not predict the target value much better than a random function. However, since these trees are built sequentially, upon each other's loss gradients, they aggregate the predictive power of the model and perform much better as an ensemble.

The hyperparameters for this regressor are very similar to the random forest's. There are still variables for the minimum number of samples at a leaf, the number of estimators n, the maximal number of features to consider for a decision, and the maximal depth of the estimators. However, there are other hyperparameters that affect the regressor. Since the estimator is based on the gradient of a loss function, the choice of this function is central to its performance. For regression, the available loss functions were least-squares, the least absolute deviation, the Huber loss, and the Quantile loss. Finally, there is a parameter for the learning rate, which lowers the contribution of each individual tree the more trees there are.

Stochastic Gradient Descent Regression

This regression is a type of stepwise regression. The gradient of the loss is estimated one sample at a time and the model is updated according to its learning rate[14][15], which decreases for each sample learned from[17]. This is also a linear model with both L1 and L2 regularization[16]; as such, it can implement either one, or a combination of both, similarly to ElasticNet. Like the ElasticNet model, there is a hyperparameter α, and a hyperparameter L_1 to control the ratio between L1 and L2 regularizations. The *loss* function used is also an important hyperparameter. This loss function can also be influenced by a hyperparameter ε, if it is not the OLS loss. Finally, the learning rate η and its change schedule controls how quickly the regressor learns. The learning rate can be selected to start at a certain value η_0 and then follow different schedules, which can also be influenced by an exponent hyperparameter *pow_t*.

Available to us were four different loss functions: the OLS loss, Huber loss, ε-insensitive, and squared ε-insensitive. The OLS loss performs as above. Huber loss modifies OLS in that, past a distance of ε, the loss function changes from squared to linear. This makes Huber loss less susceptible to outliers when compared to OLS loss. ε-insensitive ignores errors under a threshold value of ε, with a linear loss for anything further than the threshold. Finally, the squared ε-insensitive operates similarly to ε-insensitive, but uses a squared loss instead of a linear loss. For the latter three loss functions, the choice of ε is vital for the model's results.

For the learning rate schedule, the choice was between another four options: constant, adaptive, optimal, or inverse scaling. The constant schedule keeps η at the constant value η_0 throughout the model's training. The adaptive schedule similarly keeps η at the constant η_0, up until the loss difference between training iterations is too small; η is then divided by 5, and the training continues. The optimal schedule is based on α, as well as the number of samples and current iteration number and a third parameter t_0, which is an initial learning rate scaled by α. The inverse scaling schedule is based on a choice of η_0, the number of samples and current iteration number, and the exponent *pow_t*.

By the nature of following the loss gradient, this estimator is susceptible to local minima and is not guaranteed to find the global loss minimum.

Support Vector Regression

The support vector regression[12] bases itself on the assumption that there is always a tolerable amount of error. The estimator finds a subset of datapoints that substantially influence the regression estimator. Any datapoints at a maximal distance of ε to the model prediction are ignored, with the remaining datapoints being termed the support vectors. Thus, it finds a hyperplane with a maximal error of ε. This hyperplane can be calculated using different *kernels*, the most common of which are a simple linear, a polynomial, a radial basis function, or a sigmoid kernel. The estimator finds the smoothest possible hyperplane in the data it is given; however, it's not always possible for this estimator to be entirely smooth. The hyperparameter C trades off smoothness of fit with accuracy of edge cases. A last hyperparameter γ controls how much influence a single datapoint has, with a larger γ signifying lower influence over distant datapoints.

Bayesian Ridge Regression

A Bayesian ridge regression bases itself on an OLS regression and on Bayesian inference[10][11]. The model assumes that the output is distributed in a Gaussian fashion over the dimensions. The computation method itself starts with coefficients for each dimension that are equivalent to uninformative Bayesian priors. The values are given from gamma distributions that are created with hyperparameter values for the shape and inverse scale. The coefficients are then influenced by the data points in a similar fashion to Bayesian inference. This model usually yields similar results to a ridge regression, except the regularization that the Bayesian ridge has is also learned from the data.

Machine Learning Methodology

In order to have consistent, reproduceable results that do not present bias, the methodology of machine learning must be followed. It is standard to separate the dataset into two: the first part for training, and the second for testing. Since most of the above estimators are prone to overfitting, that is, learning its training data very well but not being extensible, testing on data that the estimators have not been exposed to is paramount to obtaining an accurate scoring mechanism.

The separation into train and test data, and several of the estimators involve a random component. This random component is important to ensure that the estimators learn as varied data as possible - that is, so that they are not localized functions in the parameter space. However, given that it is necessary to be able to reproduce results, a state of randomness must be given to the scripts. This also fosters trust in the estimator, since it is guaranteed to always learn the same way, provided it is given the same inputs.

All of the estimators also possess *hyperparameters*, that is, parameters that must be given to the estimator and cannot be learned from the data. Examples are the L1-to-L2 regularization ratio of the ElasticNet or the number of estimators to use in a Random Forest regressor. It is not always known what the best values for these are; in order to find the best combination of hyperparameters, there are three main procedures: first, a grid search; second, a random search; and third, a pseudo-random search with gradient descent across the parameter space.

The grid search bases itself on a set list of values that must be given to the program. The program iterates over the combinations of the values given per hyperparameter until it has exhausted all possibilities. The hyperparameter combination that performed best in cross-validation is selected.

The random search bases itself on parameter distributions that must be given. For a number of iterations - also a given -, the program randomly samples from these distributions and cross-validates the model. At the end of all iterations, the best performing hyperparameter combination is selected.

The third option, a pseudo-random search with gradient descent, is similar to the random search. However, instead of sampling entirely randomly from the parameter space, it examines what the gradient is at the sampled point and finds the location in the parameter space most likely to yield good cross-validation results. This method can perform very well, but is also susceptible to local minima.

All three procedures also require what is called *cross-validation* - a separation of the training data into subsets that serve the testing purpose for finding the optimal hyperparameter combination. In our script, the training dataset was split into 10 *folds*, 9 of which were used for training and the last being used for measuring performance of the hyperparameter combination. All combinations are trained and scored, and the mean cross-validation score is established. The scoring mechanism can vary depending on the problem space, but it is usually the mean square error or the mean absolute error.

Finally, having the best hyperparameter combination, the estimator is trained on the entire training dataset. It is then given the test dataset without the target variable and predicts what it should be. The comparison between the predicted values and the true values yields the score for the estimator.

All the estimators were exposed to the same procedure. The random state of the Python code that was run was set to a constant of zero in order to ensure reproduceability of our results. 80% of the each dataset was sampled pseudo-randomly and the estimators' hyperparameter space was explored with a random search. The random search was chosen in order not to fall into local minima and due to the ease with which it could be implemented. This random search of the hyperparameter space was done for 3000 iterations, upon the end of which the best performing combination of hyperparameters was chosen. For each of the datasets, the same hyperparameter distribution was chosen. This ensures that comparisons of performance between datasets can be made confidently.

The R^2 goodness-of-fit score for each of these iterations, along with the dataset it was trained on, was measured and recorded. The individual graphs for correlations between a hyperparameter and the mean cross-validation score for that estimator can be found in the appendix. The best hyperparameter combination was then used for the following scoring procedure, which deviates from the standard machine learning procedure.

For each dataset, 1000 random shufflings of the data were created. 90% of the data was taken for training the estimators, with the remaining 10% being used for scoring. This yields a distribution of goodness-of-fit scores.

Finally, for each estimator, the goodness-of-fit distributions were compared. The *p*-values were computed between the *all* and the *atom*; between the *all* and the *all_pca*; and between the *atom* and the *atom_pca* datasets.

Results

From the above methodology, the following box plots and table were derived:

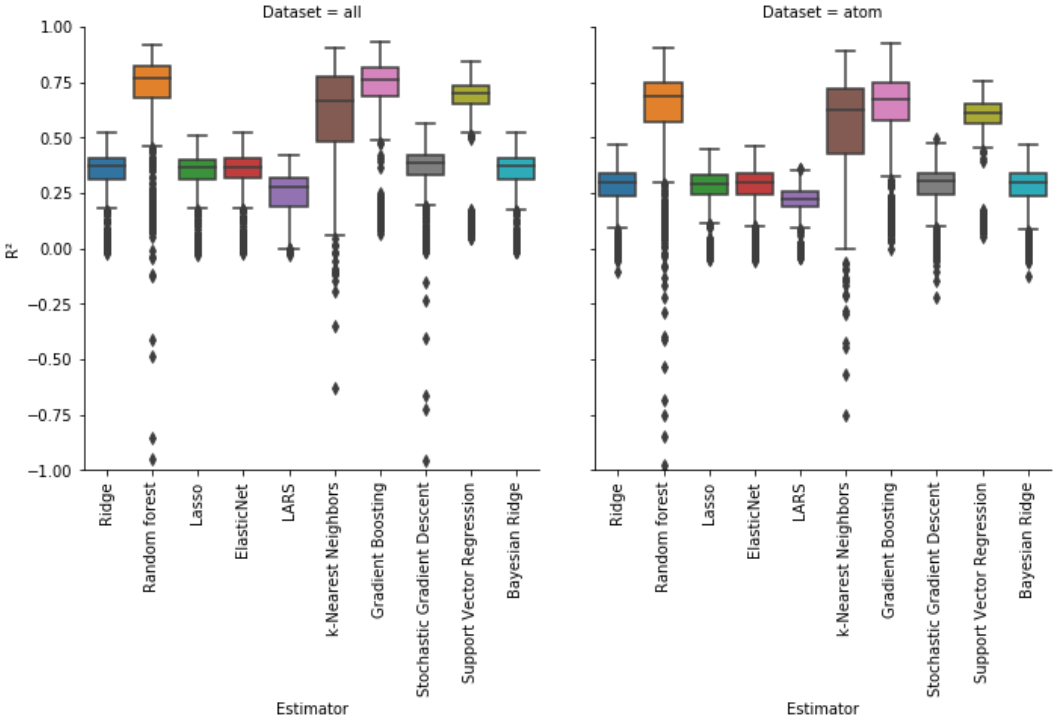


Diagram 1a: Box plot of goodness-of-fit scores by machine learning estimator and dataset. The datasets shown are *all*, which included the entire input dataset, and *atom*, which was composed exclusively of features engineered from atomic data.

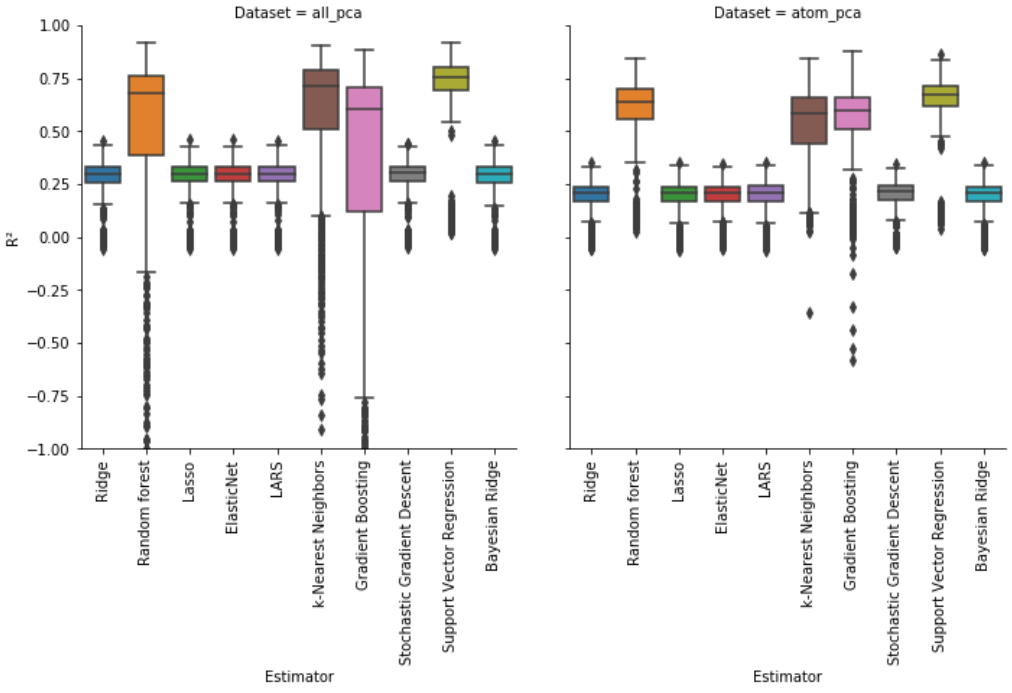


Diagram 1b: Box plot of goodness-of-fit scores by machine learning estimator and dataset. The datasets shown are *all_pca*, which is the entire input dataset having undergone a principal component analysis, and *atom_pca*, the subset of features engineered from atomic data having undergone a principal component analysis.

For clarity, the above plots were trimmed to the R^2 range of (-1, 1).

Table 2: Estimators and the *p*-values between datasets.

Estimator	*p* - all/atom	*p* - all/all_pca	*p* - atom/atom_pca
Ridge	$1.55 \cdot 10^{-41}$	$5.82 \cdot 10^{-37}$	$2.34 \cdot 10^{-74}$
Random Forest	$3.81 \cdot 10^{-11}$	$3.72 \cdot 10^{-30}$	$5.45 \cdot 10^{-01}$
Lasso	$1.58 \cdot 10^{-40}$	$4.38 \cdot 10^{-32}$	$1.49 \cdot 10^{-74}$
ElasticNet	$1.45 \cdot 10^{-41}$	$1.30 \cdot 10^{-35}$	$6.43 \cdot 10^{-79}$
LARS	$1.92 \cdot 10^{-22}$	$3.81 \cdot 10^{-10}$	$8.30 \cdot 10^{-08}$
k-Nearest Neighbors	$4.45 \cdot 10^{-05}$	$2.52 \cdot 10^{-01}$	$2.17 \cdot 10^{-02}$
Gradient Boosting	$1.66 \cdot 10^{-04}$	$4.57 \cdot 10^{-41}$	$9.82 \cdot 10^{-04}$
Stochastic Gradient Descent	$8.50 \cdot 10^{-39}$	$8.18 \cdot 10^{-38}$	$4.91 \cdot 10^{-73}$
Support Vector Regression	$3.21 \cdot 10^{-25}$	$4.11 \cdot 10^{-09}$	$3.78 \cdot 10^{-13}$
Bayesian Ridge	$6.79 \cdot 10^{-43}$	$2.71 \cdot 10^{-37}$	$2.54 \cdot 10^{-72}$

The best-performing estimator, as per mean R^2 score across all four datasets, was the support vector regression, with an average R^2 of 0.6351. The worst-performing was the LARS regression, with an average R^2 of 0.2288. The estimator with the best maximum performance was the gradient boosting regressor, with an average maximum R^2 of 0.9030. The estimator with the worst maximum performance was also the LARS regression, with an average maximum R^2 of 0.3977. It should be noted that these goodness-of-fit scores are calculated with the normalized data. The below table shows a few sample predictions de-normalized and their true values, as calculated by a k-Nearest Neighbors estimator with an R^2 score of 0.6935:

Table 3: Sample predictions from a k-Nearest Neighbors algorithm and their deviation.

InChI key	Predicted IDAC slope	Actual IDAC slope	Δ (%)
UNFUYWDGSFDHCW-UHFFFAOYSA-N	-897.32	-737.78	21.62
FAIAAWCVCHQXDN-UHFFFAOYSA-N	-144.80	-148.95	-2.78
ZNQVEEAIQZEUHB-UHFFFAOYSA-N	0.030	0.053	-43.13
DFNYGALUNNFWKJ-UHFFFAOYSA-N	0.027	0.012	124.80
LQRLKPVYXRVAPK-UHFFFAOYSA-N	-211.34	-1108.31	-80.93

Discussion

For the discussion of the results, we will start with the between-estimators performance across datasets. We will then move on to the performance between datasets, and finally, the absolute performance, once the predictions have undergone the inverse power transformation.

It is apparent from the nature of the problem at hand that the IDAC is not a linear combination of any of the engineered features. Even with the electronic patch data from COSMO-SAC, to calculate the IDAC a "decision tree" pattern is required. As such, it was expected that the linear models would perform very poorly. This was reflected in the results of the experiment: All of the linear estimators had mean goodness-of-fit scores in the 0.20-0.30 range. This tells us that while there is some variability in the data to be explained by a linear model, it is not nearly enough to fully predict the normalized IDAC. Out of all the linear models, LARS had the worst average performance. This is likely due to the large dimensionality of the data and the presence of noise, despite our removal of outliers. The three best linear models - the Ridge regression, the ElasticNet and the Stochastic Gradient Descent - had marginally better performances. This points us to regularization being an important factor in these linear models - specifically, L2 regularization. In *all*, *atom* and *atom_pca*, The best hyperparameter combination for ElasticNet, in fact, had very low L1 influence. The best hyperparameter combination for SGD agreed with ElasticNet: in all datasets except for *all_pca*, the regularization penalty was entirely based on L2.

As for the non-linear models, they performed far better on average. Particularly promising were both of the decision tree models, Random Forest and Gradient Boosting. The best hyperparameter combinations for Random Forest indicate that mean absolute error was the better loss scoring mechanism for the datasets having undergone PCA. For the Gradient Boosting regressor, it is apparent that far more estimators are required in order to have a proper prediction from it, especially for the PCA datasets. The k-Nearest Neighbors model also performed strikingly well, but unfortunately it cannot provide us with any information for data far away from its training data, given the nature of the model. It is possible that, given more data to train on, the k-Nearest Neighbors model would perform a lot better. Finally, the Support Vector model was also particularly promising: It provided a very tight distribution of goodness-of-fit scores, indicating that it was not as susceptible to sampling effects between the training and test datasets. The best hyperparameter combination for it was also fairly consistent across all datasets: it preferred the radial basis function kernel, as well as a C penalty hyperparameter of 2, indicating that it placed a higher priority on avoiding being extremely far from the true value in the training data. This also means the Support Vector model naturally requires a longer time to train, since it prefers a harder margin for its hyperplane.

Between datasets, it is apparent from the box plot above that the estimators trained on the PCA datasets perform much worse than their non-PCA counterparts. The most striking dropoff in accuracy is in the Random Forest and the Gradient Boosting models, which lose substantial performance on the *all_pca* dataset. Given that they are both based on decision trees, this raises the possibility that the 25% variance defined by the dimensions thrown out in the PCA process play a substantial role in the leaf nodes of the decision trees. As expected, the dataset with the best overall performance was indeed *all*; this makes intuitive sense, since it is the one that contains the most information overall. LARS was the only model with an arguably worse performance in the *all* dataset, and this is once again likely due to the high dimensionality and collinearity of the data.

As for the p-values between datasets, it is also immediately clear that these are very distinct goodness-of-fit distributions. This is also what was expected, since the COSMO-SAC patch data provides so much more information on the electronic distributions of molecules. One such example would be isomers: all isomers for a molecular species would have a vastly different electronic distribution, yet mostly similar atomic statistics. The added value of this information is a great differentiator between the *all* and the *atom* datasets. The distinctness of distributions between *all* and *all_pca*, and *atom* and *atom_pca* respectively, is also clear from the outset: the latter possess only 75% of the variability in data when compared to the former. With such results, it is abundantly clear that all the models relied on the additional features for their predictions.

The one notable, curious exception to the above is the *p*-value for the Random Forest between the *atom* and *atom_pca* datasets. Even after a sample size of 1000, there is still a 54.5% likelihood that these are the same distributions for goodness-of-fit scores. While other *p*-values are in the range of 10^{-10} or lower, meaning it is extremely unlikely that they are from the same distribution, this outlier stands out. It is possible that this is only an artifact of the vast array of comparisons we are making in this paper - if you explore different hypotheses repeatedly, it is statistically likely that 5% of the analyses will have a *p*-value of 0.05 -, but it is interesting nonetheless.

If we see these *p*-values as indicators of mechanisms through which an estimator learned, we can also glean additional information. It is clear that k-Nearest Neighbors learns through approximately the same mechanism every time, except its distance estimates are more precise the more dimensions we have in the dataset. This shows in its *p*-values: while the distributions are obviously very different, they are still similar enough to yield *p*-values that are understandable. This contrasts greatly with the Stochastic Gradient Descent *p*-values: these distributions are so different that one could surmise they have learned different mechanisms of interpreting the data they are given.

However, goodness-of-fit scores can only go so far in displaying accuracy. For this purpose, we have taken some of the predictions of a k-Nearest Neighbors model and undergone the inverse Yeo-Johnson transformation in order to compare the predicted and actual IDAC slopes in absolute terms. Table 3 shows us a small sample of predicted values and their deviations from the actual values for an estimator that had a goodness-of-fit score of 0.6935. While the predicted values are obviously not on the mark, they are still within one order of magnitude for this sample, which is a good indicator. It is likely that such a predictor could be used as a first, very non-specific filter for estimating IDAC slopes around the temperature of 300 K. However, it would still require a lot of experimentation and research in order to establish the true value of the IDAC at these temperatures. If we took the estimator that performed best in exclusively atomic data - namely, the Gradient Boosting regressor - and further optimized it, it is within the realm of possibility that we could find a better set of hyperparameters for the estimator. If we go one step further and take the predicted values for several estimators, such as in Table 4,

Table 4: Sample predictions from four different estimators, their average prediction, and their deviation.

InChI key	k-Nearest Neighbors prediction	Gradient Boosting prediction	Random Forest prediction	Support Vector prediction	Average prediction	Actual IDAC slope	Δ (%)
QWVGKYWNOKOFNN-UHFFFAOYSA-N	-5.36	-4.52	-1.48	4.95	-1.60	-3.15	-49.05
KVZJLSYJROEPSQ-OCAPTIKFSA-N	$-2.37\cdot10^{4}$	$-2.42\cdot10^{4}$	$-2.01\cdot10^{4}$	$-1.03\cdot10^{4}$	$-1.96\cdot10^{4}$	$-2.65\cdot10^{4}$	-26.11
ARCGXLSVLAOJQL-UHFFFAOYSA-N	-0.73	-74.43	-69.41	-116.76	-65.33	0.00753	$-8.7\cdot10^{5}$
LWCSGONZUGPMHI-UHFFFAOYSA-N	-52.61	-349.75	-101.17	-220.58	-181.03	-6.17	2832.4
AFYPFACVUDMOHA-UHFFFAOYSA-N	-323.20	-336.86	-182.53	16.00	-206.65	-62.74	229.39

it is likely that we will have a moderate understanding of the IDAC slope at 300 K - enough to inform us whether a molecular species likely has a good slope for further experiments.

Conclusion/Recommendations for Future Investigations

The impact of machine learning on the field of thermodynamics, and on predicting thermodynamic properties, has not yet shown itself fully. This paper is proof that simple machine learning algorithms are able to learn a modest amount of information from relatively little data, enough to provide predictions for the IDAC slope that are mostly within one order of magnitude of the actual value. Despite the accuracy of these predictions not being superb, it is clear that they are better than a random guess - and if given more data points and given greater exploration of the hyperparameter space for each of these estimators, it is natural that they will predict the IDAC far better than what this paper presents.

As such, future research on this topic should focus on the non-linear estimators: k-Nearest Neighbors, a highly dataset size dependent estimator, Random Forest, a simple yet strikingly accurate model, and Gradient Boosting and Support Vector regressions, both estimators very dependent on training time. The research should acquire far more datapoints, and from a more varied parameter space, in order to be more generalizable; and it should aim to improve the performance of these models when given exclusively atomic data. If such a model were available, it would certainly compete with UNIFAC and COSMO-SAC, at least in predicting the IDAC and its slope.

Another possibility includes the creation of a neural network model to explore this space. Such models should learn properties directly from the molecular structure of a molecular species. These models could then also create structures which present the wanted properties. This would be an entirely new paradigm of molecule and mixture creation, where we explore the possibilities of our mixtures through simple, artificial intelligence simulations first, then move on to experimenting in the lab.

On a broader scope, other thermodynamic properties should also be explored. The nuances of a molecule are very hard to capture in a model such as COSMO-SAC, but with a kind learning field such as chemistry - where interactions are immediately effective and there is feedback on whether an estimator is correct or not -, we believe these machine learning models can learn these small differences in behavior and predict thermodynamic properties far more accurately than this first foray.

Acknowledgements

I would first like to thank Jadran Vrabec, for taking my project on. I would like to thank Ian Bell and Andreas Jäger, for their help in acquiring the COSMO-SAC data I used in this work. I would also like to thank my friends and family for their support during both this thesis project and my undergraduate studies in general. Finally, I would like to thank Tobias Fischer and Bruno Kosinski, without whom I would not have been able to begin my studies in the first place.

References

(1) Yeo, I.-K. A new family of power transformations to improve normality or symmetry. Biometrika 2000, 87, 954–959.
(2) Willoughby, R. A. Solutions of Ill-Posed Problems (A. N. Tikhonov and V. Y. Arsenin). SIAM Review 1979, 21, 266–267.
(3) Santosa, F. and Symes, W. W. Linear Inversion of Band-Limited Reflection Seismograms. SIAM Journal on Scientific and Statistical Computing 1986, 7, 1307–1330.
(4) Tibshirani, R. Regression shrinkage and selection via the lasso: a retrospective. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 2011, 73, 273–282.
(5) Liu, M. and Vemuri, B. C. A Robust and Efficient Doubly Regularized Metric Learning Approach. In Computer Vision – ECCV 2012, Springer Berlin Heidelberg, 2012, 646–659.
(6) Friedman, J., Hastie, T., and Tibshirani, R. Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software 2010, 33.
(7) Zou, H. and Hastie, T. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 2005, 67, 301–320.
(8) Tibshirani, R., Johnstone, I., Hastie, T., and Efron, B. Least angle regression. The Annals of Statistics 2004, 32, 407–499.
(9) Hesterberg, T., Choi, N. H., Meier, L., and Fraley, C. Least angle and ℓ 1 penalized regression: A review. Statistics Surveys 2008, 2, 61–93.
(10) Tipping, M. E., and Smola, A. (Ed.). Sparse Bayesian learning and the relevance vector machine. Journal of Machine Learning Research 2001, 1, 211-244.
(11) MacKay, D. J. C. Bayesian Interpolation. Neural Computation 1992, 4, 415–447.
(12) Drucker H., Burges C. J., Kaufman L., Smola A. J., and Vapnik V., Support vector regression machines. Advances in neural information processing systems 1997, 155-161.
(13) Pedregosa F. et al., Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 2011, 12, 2825-2830.
(14) Bottou L., and LeCun Y. Large Scale Online Learning, Advances in Neural Information Processing Systems 16 (NIPS 2003), 2004.
(15) Zhang T. Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms, In Proceedings of ICML '04, 2004.
(16) Tsuruoka Y., Tsujii J., and Ananiadou S. Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty, In Proceedings of ACL-IJCNLP, 2009, 477-485.
(17) Shalev-Shwartz S., Singer Y., and Srebro N. Pegasos: Primal estimated sub-gradient solver for SVM, In Proceedings of ICML '07, 807-814.
(18) Altman, N. S. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. The American Statistician 1992, 46, 175–185.
(19) Bentley, J. L. Multidimensional binary search trees used for associative searching. Communications of the ACM 1975, 18, 509–517.
(20) Omohundro S. Five Balltree Construction Algorithms, Berkeley: International Computer Science Institute, 1989.
(21) Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. Machine Learning 2006, 63, 3–42.
(22) Breiman, L. Machine Learning 2001, 45, 5–32.
(23) Ho, T. Random decision forests, Proceedings of the 3rd International Conference on Document Analysis and Recognition, 1, 278.

(24) Friedman, J. H. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 2001, 29, 1189–1232.
(25) Friedman, J. H. Stochastic gradient boosting. Computational Statistics & Data Analysis 2002, 38, 367–378.
(26) Ceriotti, M. Unsupervised machine learning in atomistic simulations, between predictions and understanding. The Journal of Chemical Physics 2019, 150, 150901.
(27) Krizhevsky, A., Sutskever, I. and Hinton, G. E. Imagenet classification with deep convolutional neural networks, In Advances in neural information processing systems, 2012, 1097-1105.
(28) Shen, S., Jiang, H. and Zhang, T. Stock market forecasting using machine learning algorithms. Department of Electrical Engineering, Stanford University, Stanford, CA, 2012, 1-5.
(29) Cruz, J. A. and Wishart, D. S. Applications of Machine Learning in Cancer Prediction and Prognosis. Cancer Informatics 2006, 2.
(30) Sandler, S. I. Infinite dilution activity coefficients in chemical, environmental and biochemical engineering. Fluid Phase Equilibria 1996, 116, 343–353.
(31) Tse, G., Orbey, H., and Sandler, S. I. Infinite dilution activity coefficients and Henry's law coefficients of some priority water pollutants determined by a relative gas chromatographic method. Environmental Science & Technology 1992, 26, 2017–2022.
(32) Nielsen, F. , Olsen, E. , and Fredenslund, A. Henry's Law Constants and Infinite Dilution Activity Coefficients for Volatile Organic Compounds in Water by a Validated Batch Air Stripping Method. Environmental Science & Technology 1994, 28, 2133–2138.
(33) Ellis, S. R. M. and Jonah, D. A. Prediction of activity coefficients at infinite dilution. Chemical Engineering Science 1962, 17, 971–976.
(34) Fredenslund, A., Jones, R. L. and Prausnitz, J. M. Group-contribution estimation of activity coefficients in nonideal liquid mixtures. AIChE Journal 1975, 21, 1086-1099.
(35) Wang, S., Sandler, S. I., and Chen, C.-C. Refinement of COSMO−SAC and the Applications. Industrial & Engineering Chemistry Research 2007, 46, 7275–7288.
(36) Jakob, A., Grensemann, H., Lohmann, J., and Gmehling, J. Further Development of Modified UNIFAC (Dortmund): Revision and Extension 5. Industrial & Engineering Chemistry Research 2006, 45, 7924–7933.
(37) Fingerhut, R., Chen, W.-L., Schedemann, A., Cordes, W., Rarey, J., Hsieh, C.-M., Vrabec, J., and Lin, S.-T. Comprehensive Assessment of COSMO-SAC Models for Predictions of Fluid-Phase Equilibria. Industrial & Engineering Chemistry Research 2017, 56, 9868–9884.
(38) Klamt, A. Conductor-like Screening Model for Real Solvents: A New Approach to the Quantitative Calculation of Solvation Phenomena. The Journal of Physical Chemistry 1995, 99, 2224–2235.
(39) Klamt, A. and Schüürmann, G. COSMO: a new approach to dielectric screening in solvents with explicit expressions for the screening energy and its gradient. Journal of the Chemical Society, Perkin Trans. 2 1993, 799–805.
(40) Hadamard, J. Sur les problèmes aux dérivées partielles et leur signification physique, 1902, Princeton University Bulletin, 49–52.
(41) Tipping, M. E., Bishop, C. M. Probabilistic principal component analysis, 1999. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 61.

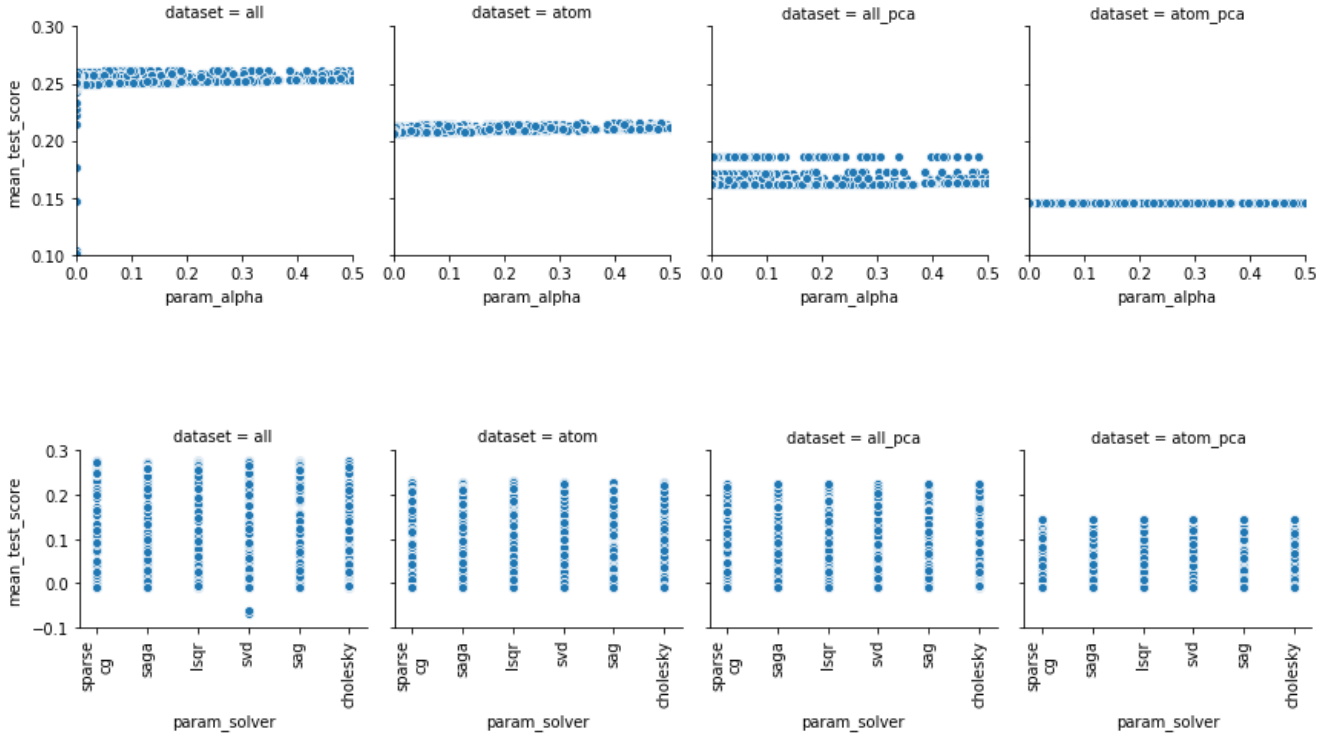
Appendices

Python Library Versions

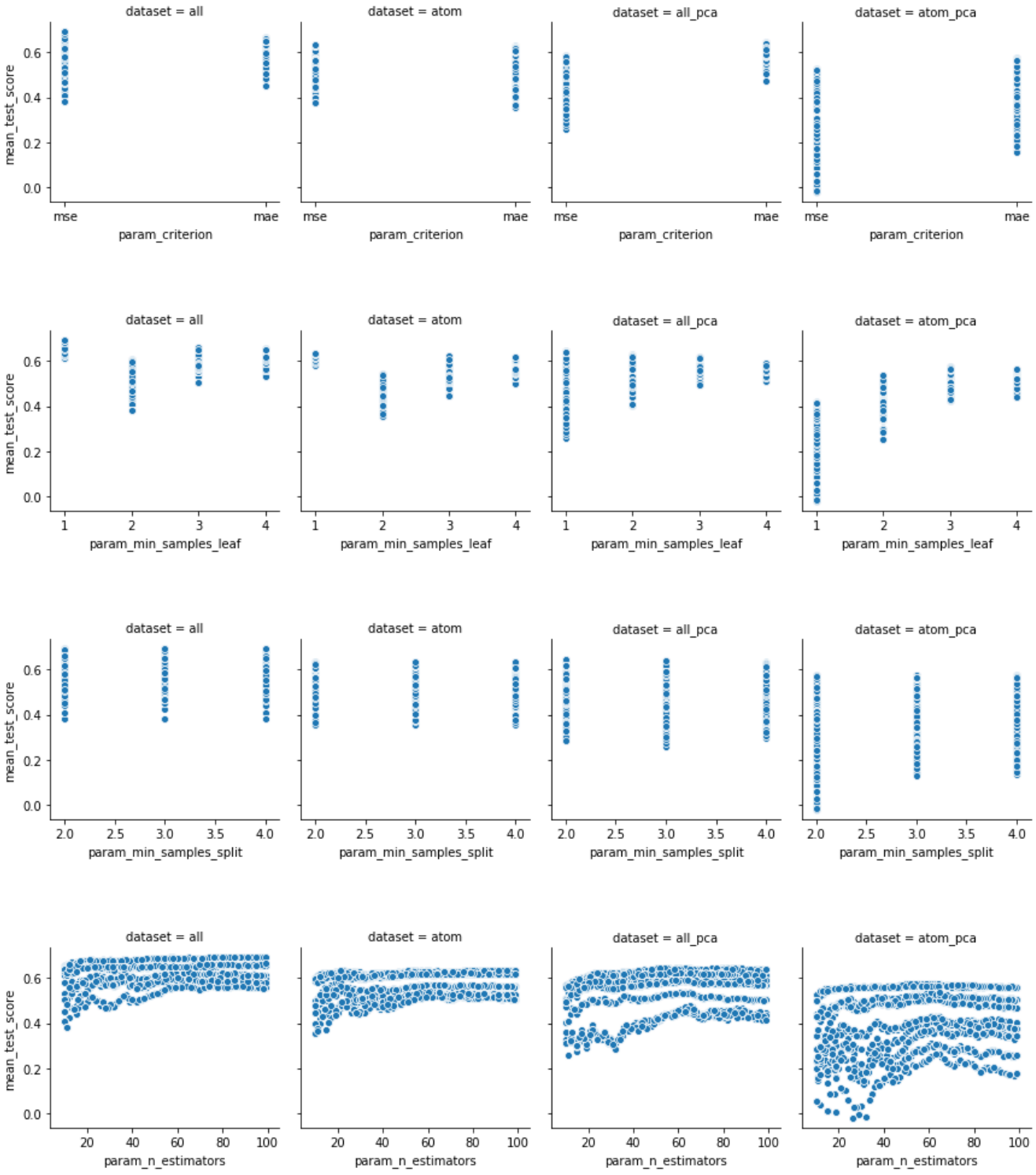
Library	Version
Python	v.3.7.0
scikit-learn	v.0.20.3
pandas	v.0.23.4
NumPy	v.1.16.0
SciPy	v.1.1.0

Hyperparameter Graphs

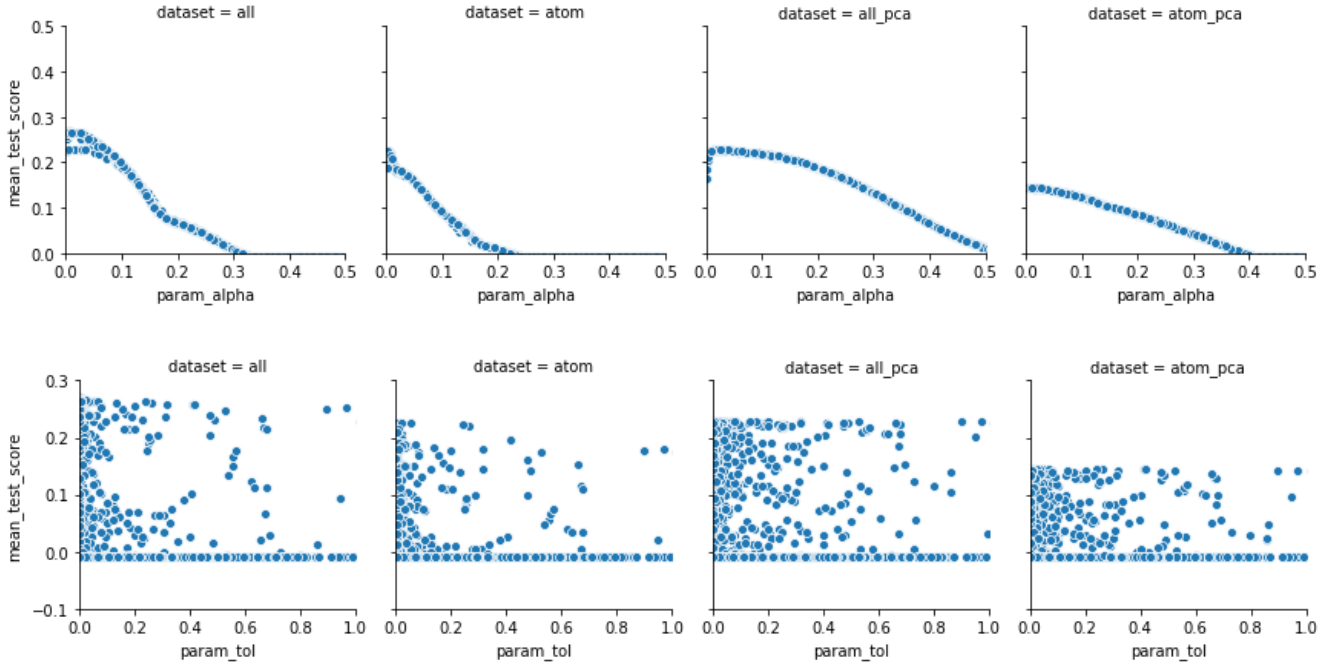
Ridge Regression



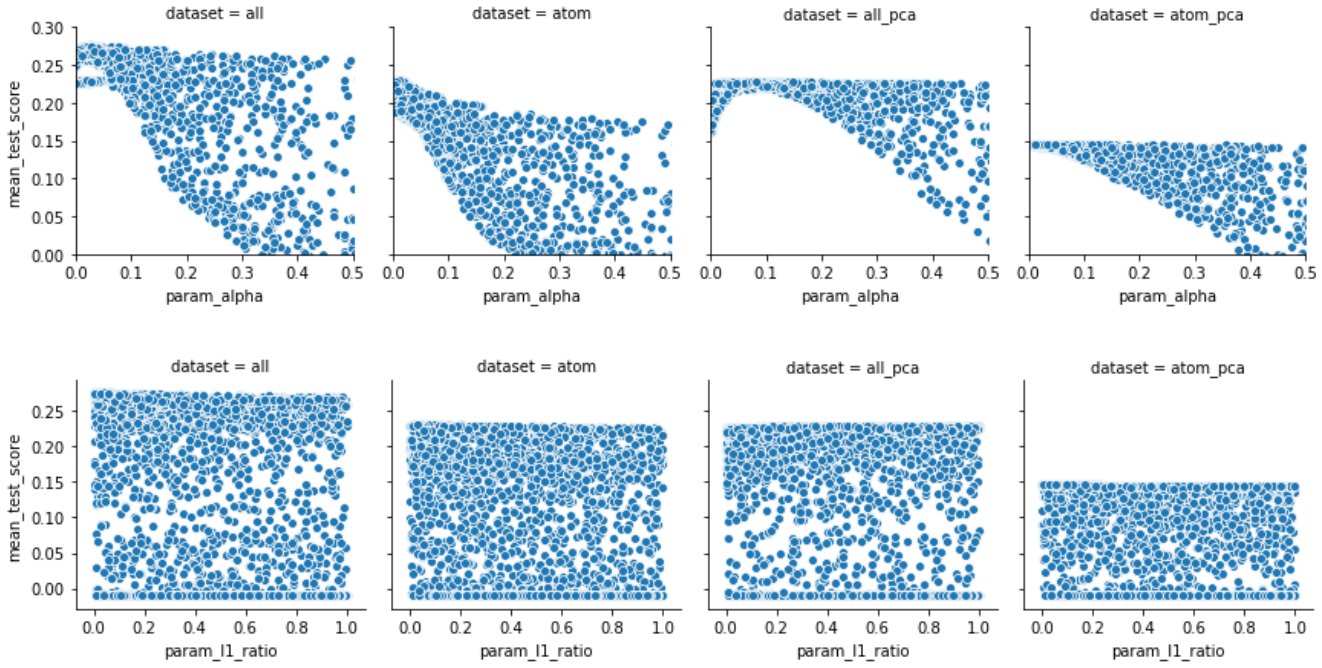
Random Forest Regression

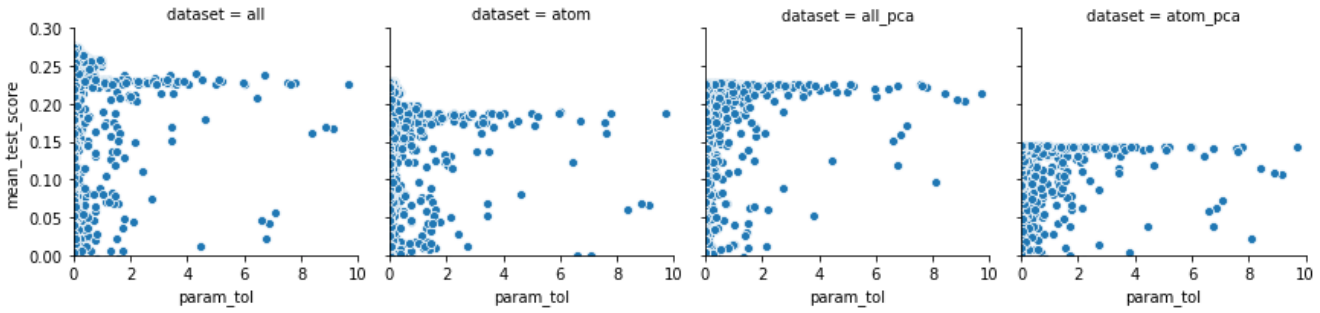

dataset = all
dataset = atom
dataset = all_pca
dataset = atom_pca
mean_test_score
param_criterion
mse
mae
param_min_samples_leaf
param_min_samples_split
param_n_estimators

Lasso Regression

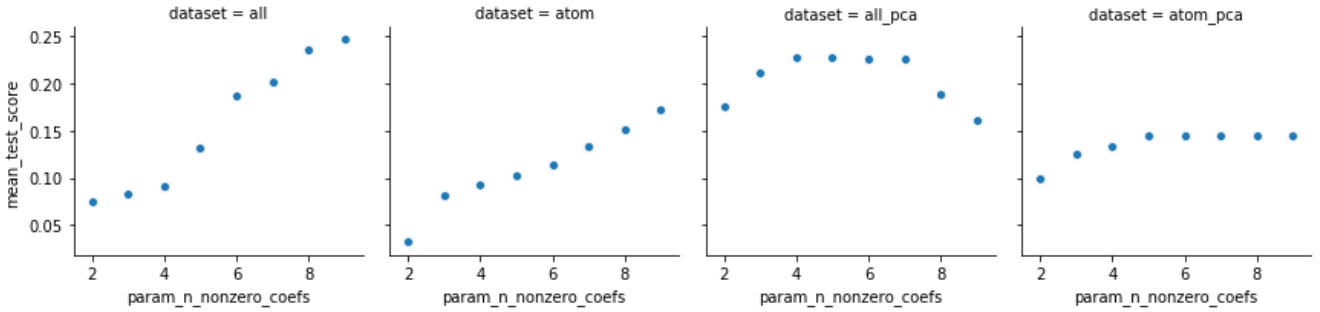


ElasticNet

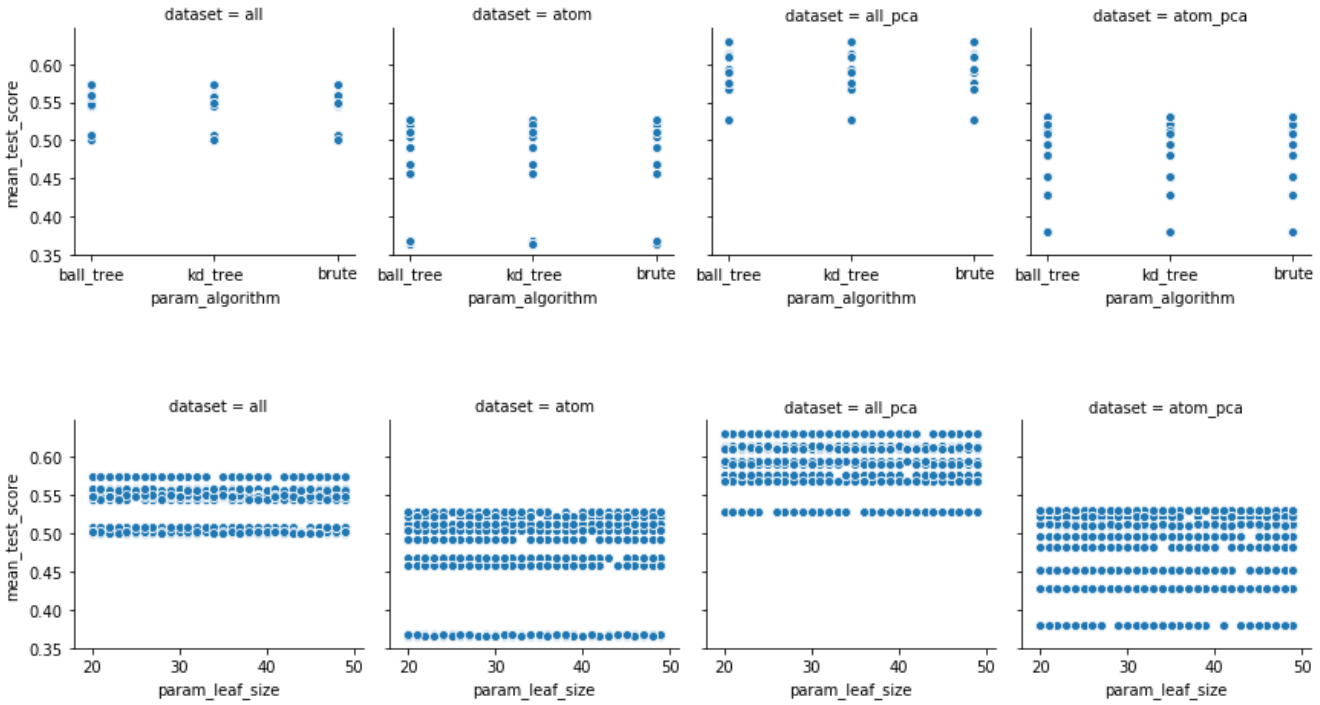


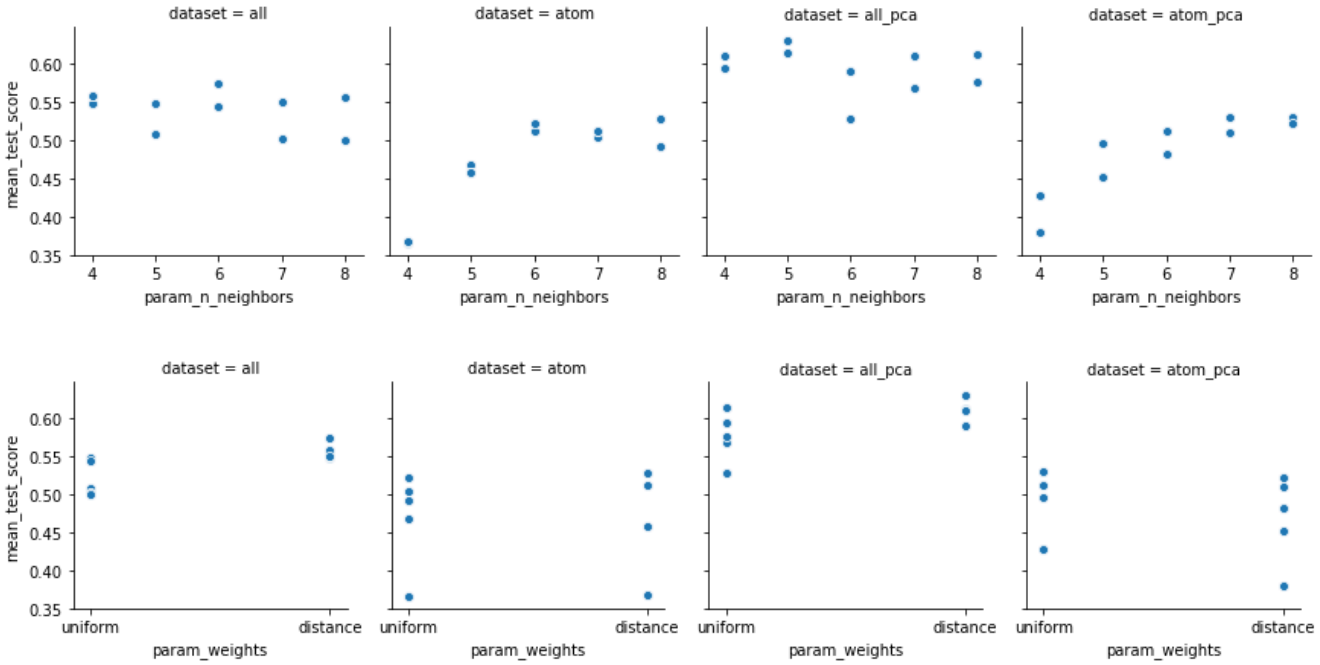


Least Angle Regression

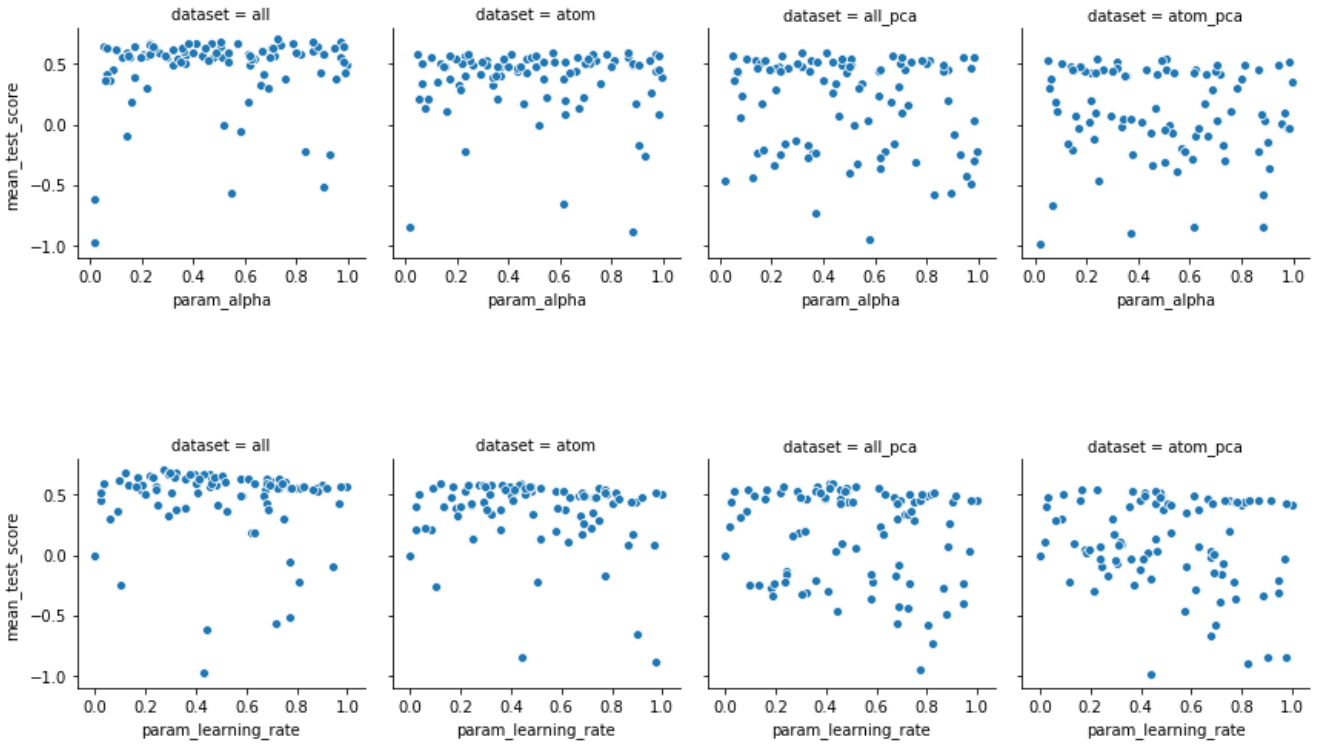


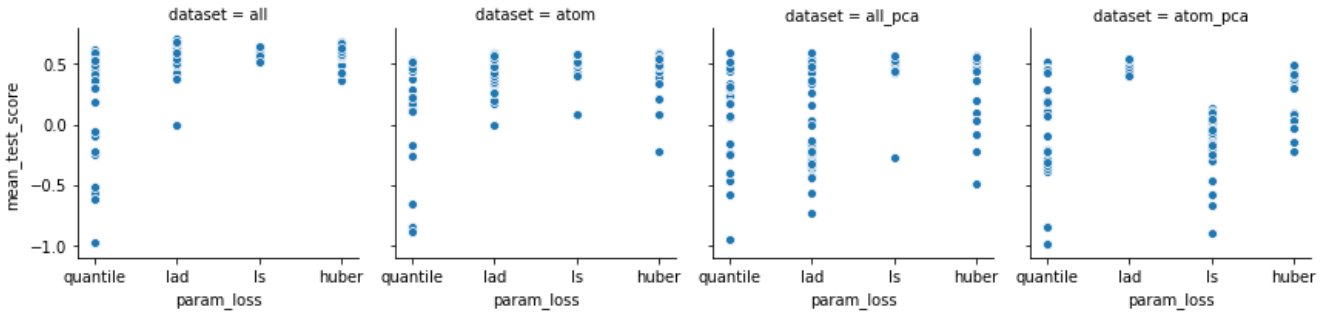
k-Nearest Neighbors Regression

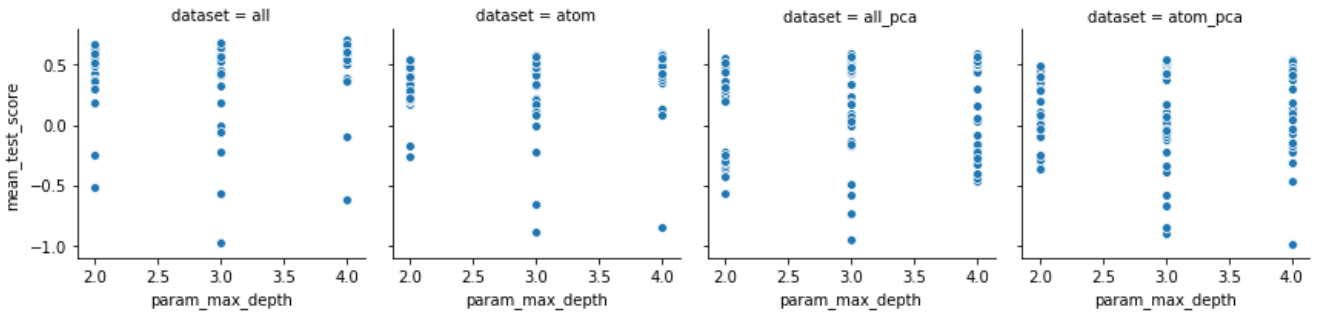


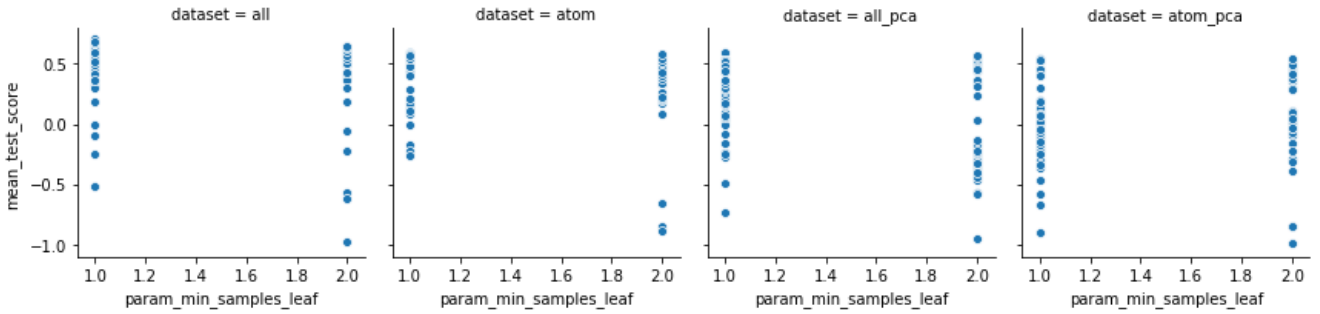


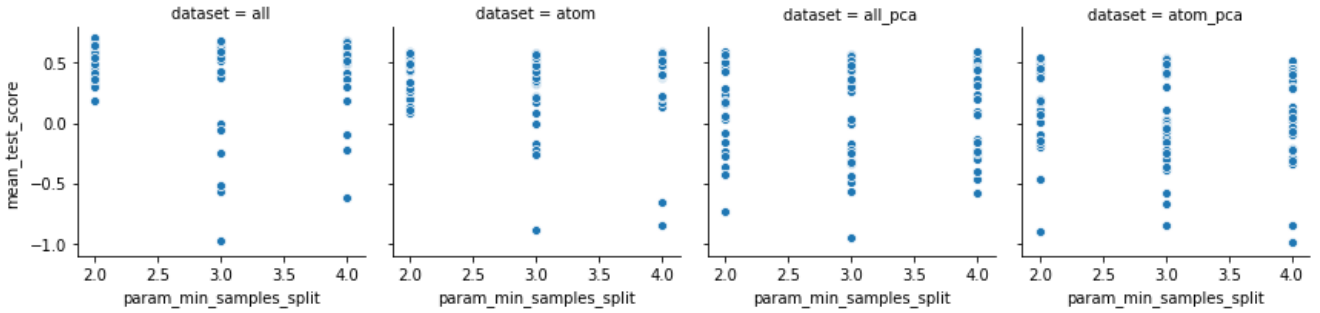
Gradient Boosting Regression

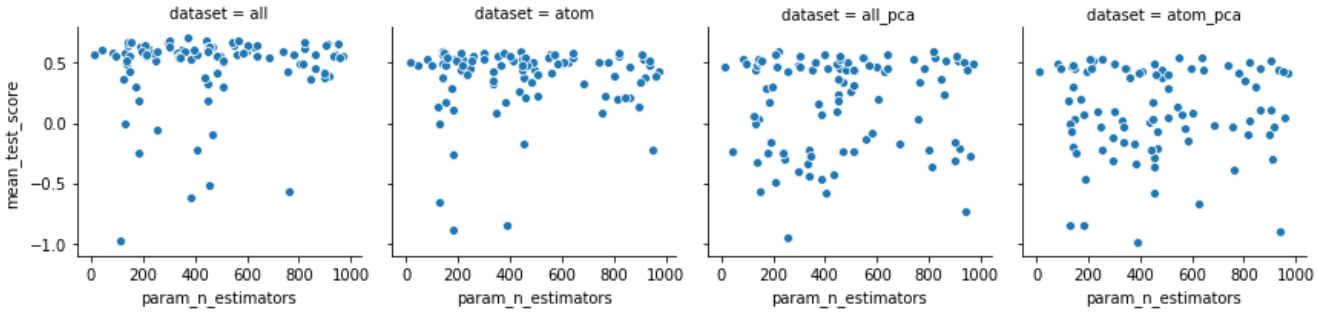



dataset = all
dataset = atom
dataset = all_pca
dataset = atom_pca
mean_test_score
0.5
0.0
−0.5
−1.0
quantile
lad
ls
huber
param_loss

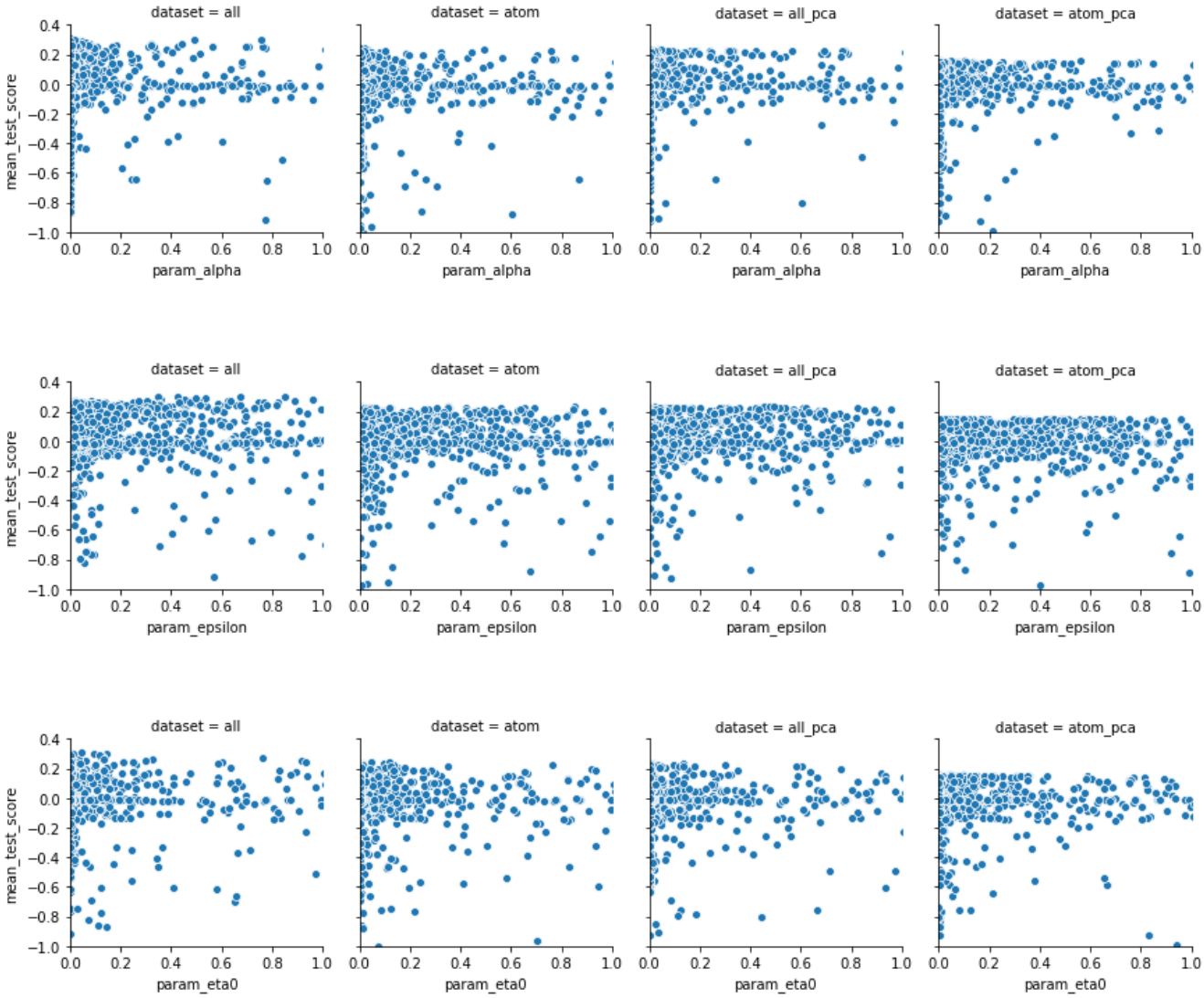

dataset = all
dataset = atom
dataset = all_pca
dataset = atom_pca
mean_test_score
0.5
0.0
−0.5
−1.0
2.0
2.5
3.0
3.5
4.0
param_max_depth

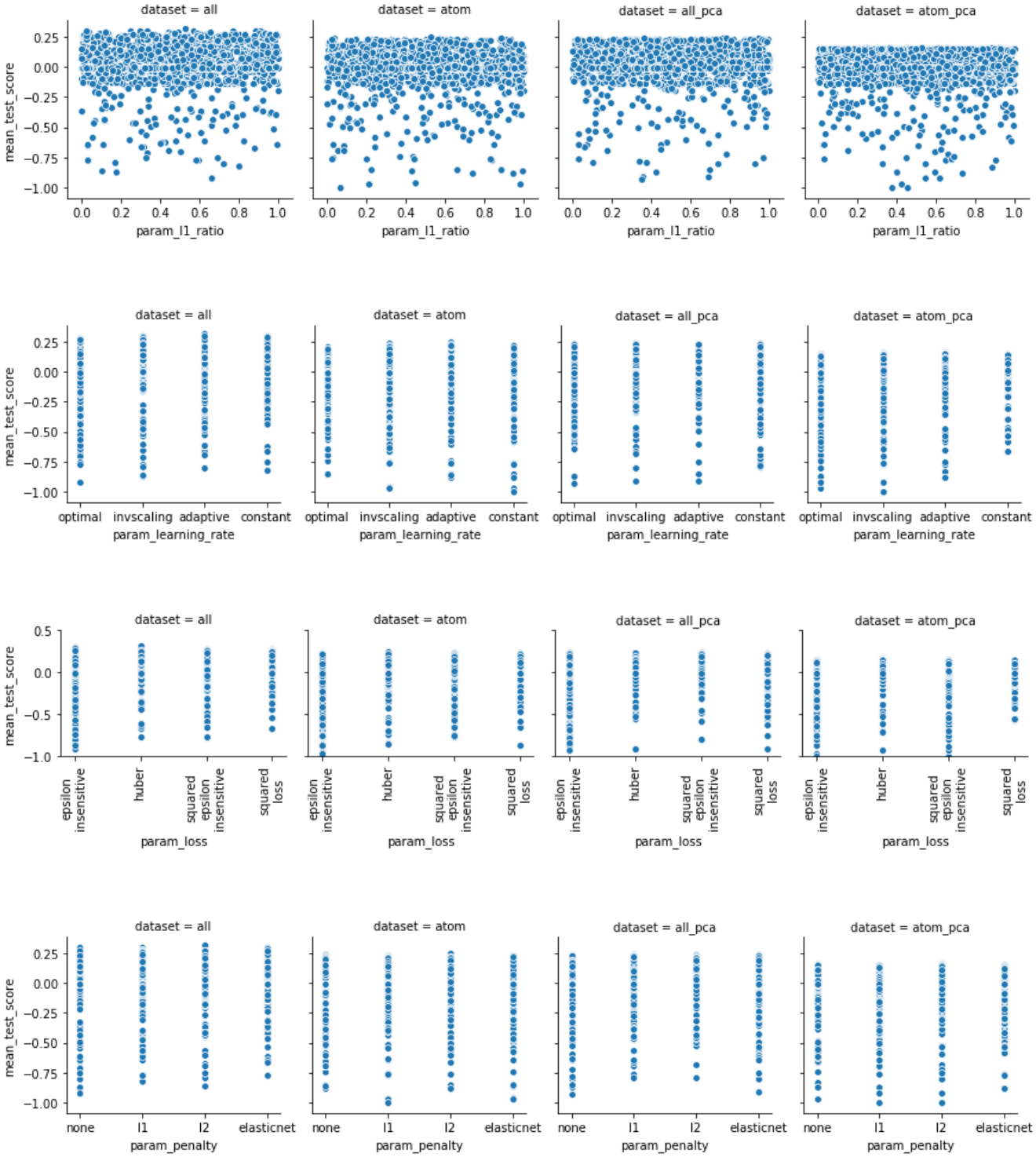

dataset = all
dataset = atom
dataset = all_pca
dataset = atom_pca
mean_test_score
0.5
0.0
−0.5
−1.0
1.0
1.2
1.4
1.6
1.8
2.0
param_min_samples_leaf

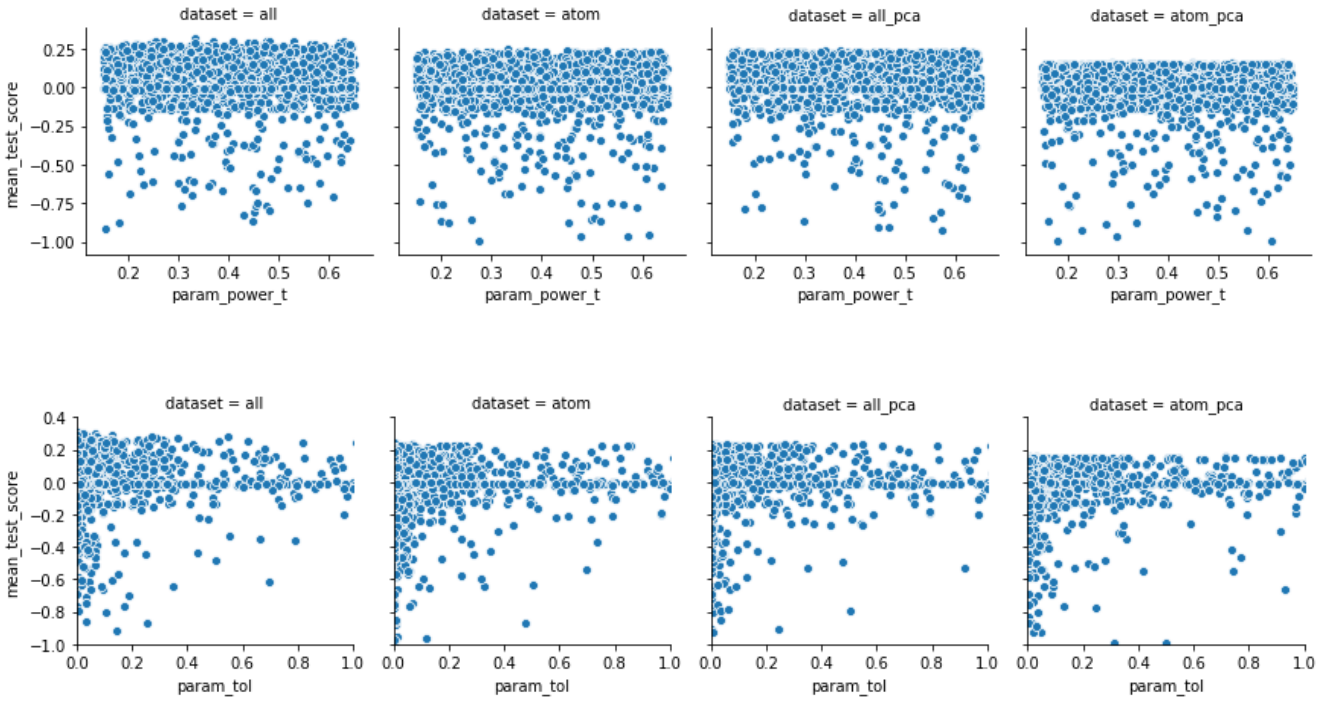

dataset = all
dataset = atom
dataset = all_pca
dataset = atom_pca
mean_test_score
0.5
0.0
−0.5
−1.0
2.0
2.5
3.0
3.5
4.0
param_min_samples_split



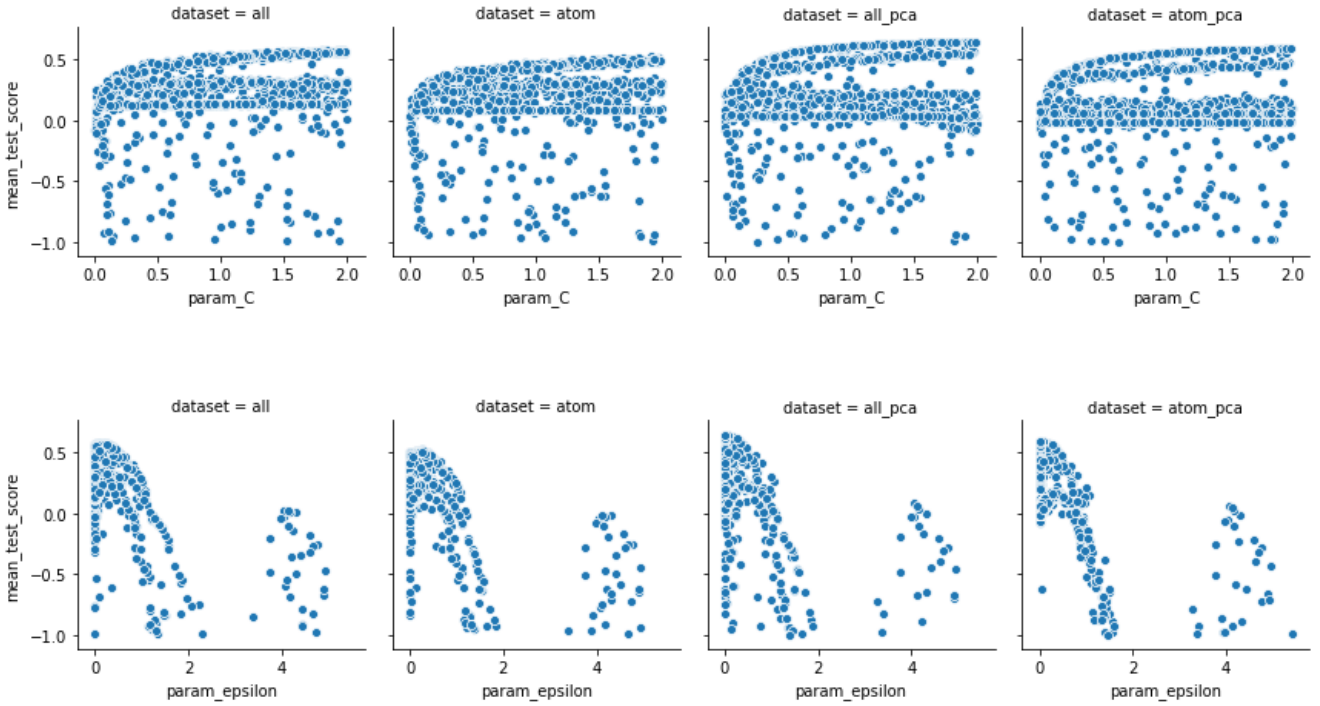
Stochastic Gradient Descent Regression

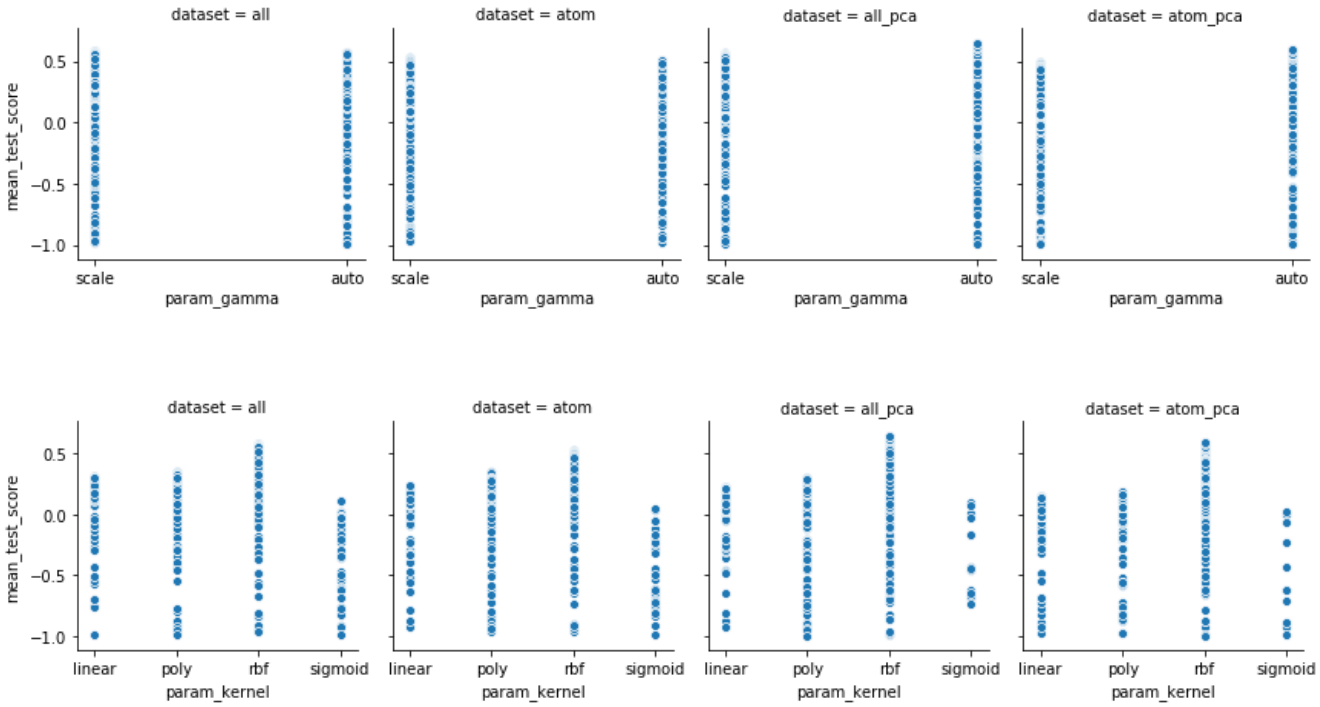



dataset = all
dataset = atom
dataset = all_pca
dataset = atom_pca
mean_test_score
param_l1_ratio
param_learning_rate
optimal
invscaling
adaptive
constant
param_loss
epsilon insensitive
huber
squared epsilon insensitive
squared loss
param_penalty
none
l1
l2
elasticnet

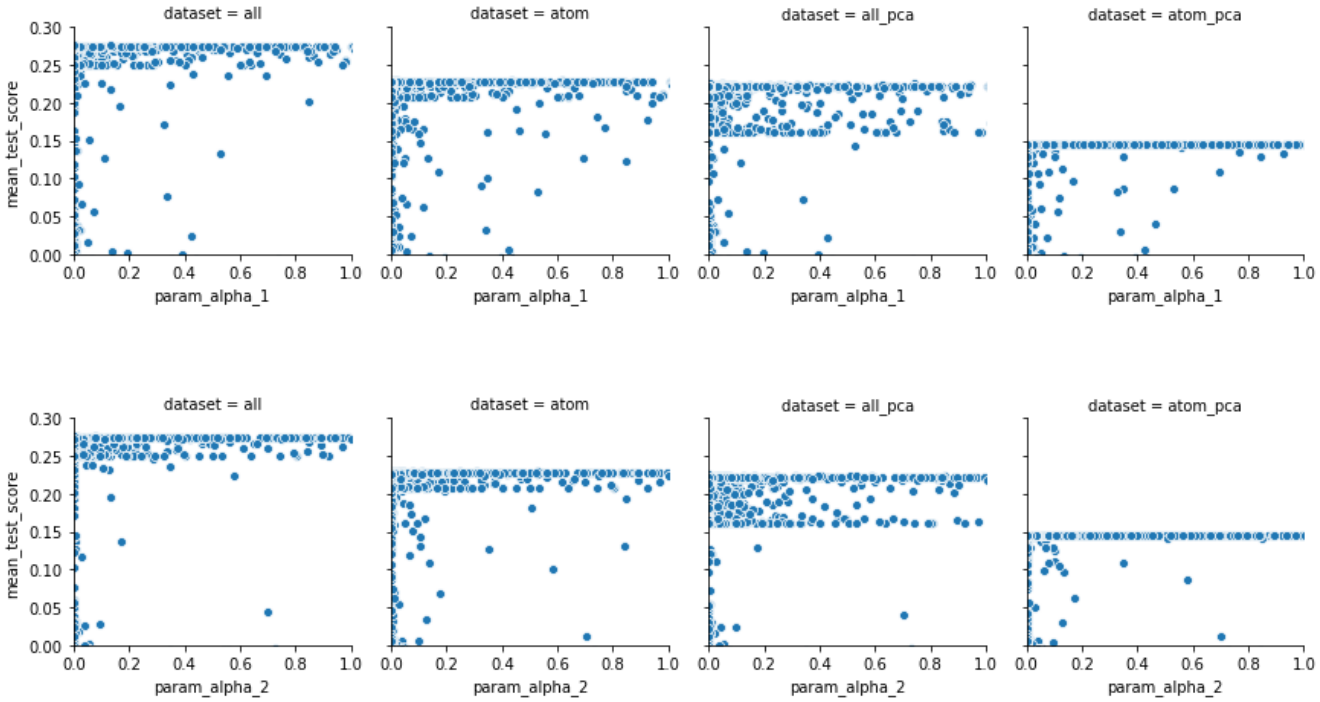


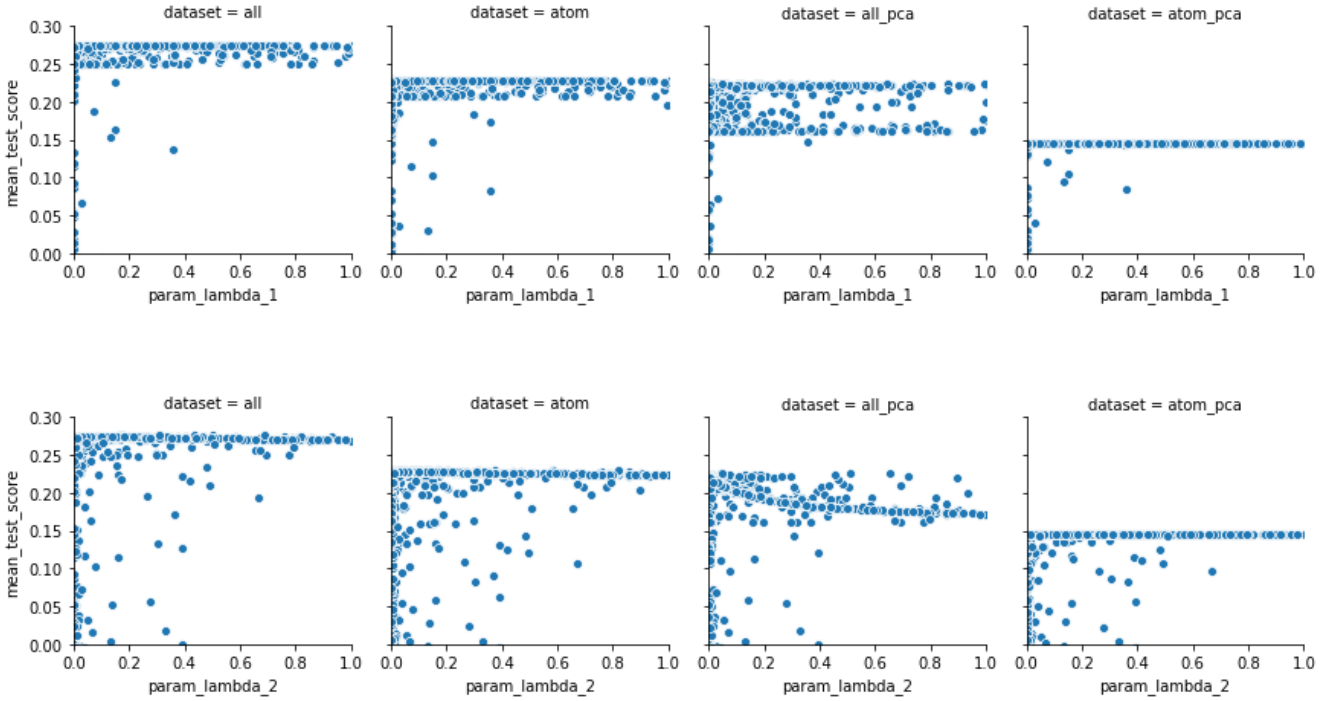
Support Vector Regression





Bayesian Ridge Regression




dataset = all
dataset = atom
dataset = all_pca
dataset = atom_pca
mean_test_score
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.0
0.2
0.4
0.6
0.8
1.0
param_lambda_1
param_lambda_2