Predicting the Heat of Formation of One-, Two-, and Three-Component Materials Using Machine Learning

Capstone Project: Final Report

Isaiah Steinke
Student ID #A01338463
MATH 688

Abstract

In materials informatics (MI), researchers utilize machine learning (ML) techniques to help discover new materials, predict key material properties, and aid in research and development. In this study, we build ML models to predict a key material property—the heat of formation—for 9,590 materials containing 1–3 elements using data from the Computational Materials Repository and features generated from the properties of the elements in the chemical formula. Our model process predicts the heat of formation to be zero for one-component materials; for two- and three-component materials, a variety of regression- and tree-based models were developed. Our best model uses boosted decision trees to achieve a mean absolute error (MAE) of 68 meV/atom, which compares favorably to other models in the literature and exceeds a benchmark of 96 meV/atom calculated from density functional theory and experiments. Given the low MAE and the low spread in the prediction errors on the test set, the boosting model will be suitable for MI tasks requiring accurate prediction of the heat of formation.

Keywords: Boosting, decision trees, heat of formation, machine learning (ML), materials informatics (MI), random forests, regression.

1. Introduction

In general, materials scientists are interested in the properties and applications of materials and utilize principles from physics, chemistry, and engineering. In particular, they are interested in the processing–structure–property–performance (PSPP) relationships of materials, as these can be used to improve current materials, discover and design new materials, and guide future research directions. However, many of these PSPP relationships are complex and are not thoroughly understood.

The field of materials science has evolved according to the four paradigms of science [1]–[3]. The first three paradigms are well-established. The first paradigm is empirical science or observation and experimentation. The second is theoretical model-based science—laws and theories in the form of mathematical equations. However, these models have become increasingly complex over time, and analytical solutions are often not readily obtained. Hence, the third paradigm was developed: computational science via simulations of complex phenomena, e.g., the use of density functional theory (DFT), molecular dynamics (MD), etc.

Since 2000, a fourth paradigm has emerged with the advent of big data, as scientific experiments and simulations now have the potential to generate large amounts of data characterized by their volume, velocity, and variety. This fourth paradigm is data-driven science using big data; in the context of materials science, it has led to the emergence of a field known as materials informatics (MI). In MI, data mining, machine learning (ML), and artificial intelligence are used to discover new materials and materials phenomena, to improve materials selection and experimentation, and to guide the development of materials in industry, government, and academia.

Given the very recent development of MI, there are still many fundamental challenges regarding the use of big data for materials science. Hill *et al.* [2] have enumerated the major challenges and discussed the current efforts to address them, such as The Materials Project [4]. Himanen *et al.* [5] discuss many of these same challenges and the roles of the open science movement, open access publishing, open access data, and open-source software in MI.

Agrawal and Choudhary [1], [3] discussed the overall workflow for MI. Their proposed workflow for knowledge discovery mines databases of materials to discover PSPP relations for materials design. In general, it follows a similar process as conventional data-science analyses:

data extraction and preprocessing, data reduction and selection, modeling and relationship mining, and evaluation. According to their review [1], popular mining techniques for MI include naïve Bayes, Bayesian networks, logistic regression, *k* nearest-neighbors (kNN), artificial neural networks (ANNs), support vector machines (SVMs), decision trees, bagging, boosting, and random forests. In [3], they focus on reviewing the use of neural networks for MI, including ANNs, convolutional neural networks (CNNs), and generative adversarial networks (GANs). In addition, they present many examples of MI: the prediction of fatigue in steel [1]; the discovery of stable compounds [1]; microstructure optimization, characterization, and design [1], [3]; and the prediction of the crystal structure and its use for predicting properties [3].

Given the workflow in [1], some researchers [6], [7] have discussed best practices for MI and highlighted some commonly encountered issues. Wagner and Rondinelli [7] discuss potential pitfalls: overfitting, the presence of underrepresented classes in classification (i.e., "imbalanced" training sets), descriptor selection, feature extraction, model interpretability, cross-validation (CV)/regularization, and model choice. In particular, the selection of appropriate descriptors and the extraction and creation of features are important and are related to the particular materials system selected and/or the desired output of the model. Moreover, composition-based feature vectors [6], [8], [9] or universal property-labeled materials fragments [10] may be used to improve modeling performance.

With this general background, we utilize ML models to predict the heat of formation of one-, two-, and three-component materials (i.e., materials with 1–3 elements in their chemical formula). Following the best practices and general workflow for MI, we have created additional features to augment the small number of variables in the original dataset to potentially improve modeling performance. After exploring this augmented dataset, ML models were trained and tested, and the performance of these models was assessed.

The heat of formation of a compound is an important fundamental thermodynamic quantity. In short, the heat of formation (also known as the enthalpy of formation) is the energy needed or given off when a compound is formed from its constituent elements. Negative values indicate that energy is given off and implies that the compound is stable. In contrast, positive values indicate that energy is needed to form the compound and that it is not stable under ambient conditions. Since no changes are needed for elemental compounds, the heat of formation of a one-component material is zero. For materials containing two or more components, the atoms comprising the material typically undergo some changes during the formation of the compound. For example, an ionic compound such as NaCl (table salt) will undergo some notable changes: the elements Na and Cl must become ionized and the atoms then form an ionic bond. As the heat of formation is used in many practical calculations and models, accurate values are important. For MI purposes, it may be useful to predict the heat of formation for unknown compounds in order to assess their viability during research and development or to estimate the costs associated with a manufacturing process.

2. Objectives

The objectives for the capstone project are as follows:

1. Use the dataset and composition-based features to find potential groups of materials to derive potential insights/patterns or to aid in the modeling process.
2. Develop ML models to predict the heat of formation of one-, two-, and three-component materials.
3. Assess the accuracy of the predictions of the heat of formation.

4. Assess the viability and performance of the models developed to predict the heat of formation and compare their performance to other models in the literature.

3. Literature Review

There are multiple studies on predicting the heat of formation using ML techniques. Some early work used a database of DFT calculations and a basic model-building framework [11]. Two models were utilized: one was based on a simple heuristic and linear regression and the other was based on ensembles of decision trees. These models only used composition-dependent features and demonstrated good results for materials discovery. Deml *et al.* [12] used linear regression to predict the heat of formation of 2,046 compounds; these models also used composition-dependent features. In their work, the best model used stepwise linear regression with 82 features and was able to achieve an average mean absolute error (MAE) of ~80 meV/atom.

Ward *et al.* [8] developed a program that creates features from the constituent elements of a compound called the Materials Agnostic Platform for Informatics and Exploration (Magpie). With these features, they utilized data from the Open Quantum Materials Database (OQMD) [13], [14] to build models using 10-fold CV. The best model used the reduced-error pruning decision tree (REPTree) with the random subspace technique, as assessed by the lowest MAE in CV: 88.2 eV/atom. As a benchmark, Kirklin *et al.* carried out an analysis of the heat of formation calculated using DFT for materials in the OQMD and compared these DFT values to experimental values [13]. The overall MAE between these values is 96 meV/atom, and the MAE falls in the range of 81–136 meV/atom depending on the details of the DFT calculations and the type of material. Hence, the MAE obtained by Ward *et al.* is a modest improvement.

Ward *et al.* later extended their work to include other features in addition to those generated by Magpie; these new features are related to the Voronoi tessellation of the crystal structure of each compound [15]. For a large dataset of 435,000 compounds from the OQMD, their random forests model was able to achieve a low MAE of 80 meV/atom. Moreover, they showed that their model can accurately estimate the heat of formation of materials that are not in the training set and identify materials with large heats of formation.

Jha *et al.* [16] developed a deep neural network (DNN) model called *ElemNet* to predict the heat of formation. Compared to the models developed by Ward *et al.* [8], [9], *ElemNet* is able to achieve superior performance in terms of speed and accuracy. Under 10-fold CV, the MAE is 50 meV/atom with a 17-layer DNN. Xie and Grossman [17] developed a crystal graph convolutional neural network (CGCNN) framework, which directly learns the properties of a material from the connections between the atoms in a material's crystal structure. Their framework was used to predict eight different material properties. For the heat of formation, the MAE was 39 meV/atom. A variant of a deep tensor neural network (DTNN) called SchNet was able to achieve a further improvement in the MAE to 35 meV/atom [18], and Chen *et al.* were able to reduce the MAE to 28 meV/atom using MEGNet, which is an ML model that uses graph networks [19].

4. Data

4.1. Dataset

The raw data were acquired from the Computational Materials Repository (CMR) [20] and were originally compiled in the OQMD [11], [12]. These raw data include the chemical formula (a character string), the volume of the unit cell (a continuous numeric variable with units of angstroms cubed), the sum of the atomic masses in the unit cell (a continuous numeric variable with units of grams per mole), the number of atoms in the chemical formula (an integer numeric

variable), and the heat of formation (a continuous numeric variable with units of electron volts per atom) for 9,590 one-, two-, and three-component materials. There were no missing values in the raw data.

The latest version of Magpie [8], [21] was used to process the raw data into 145 features related to the physical and chemical properties of these materials. A detailed description of these features is as follows:

- Stoichiometric attributes (total: 6): These attributes are related to the fractions of elements present in the chemical formula and are based on L^p norms, with p = 0, 2, 3, 5, 7, and 10.
- Elemental-property-based attributes (total: 132): These attributes are based on statistics of elemental properties: the minimum, maximum, range, fraction-weighted mean, average deviation, and mode. These statistics are calculated for the atomic number; Mendeleev number; atomic mass; melting temperature; column and row in the periodic table; covalent radius; electronegativity; numbers of *s*, *p*, *d*, and *f* valence electrons; total number of valence electrons; numbers of unfilled *s*, *p*, *d*, and *f* states; total number of unfilled states; and specific volume, band gap energy, magnetic moment (per atom), and space group number of the 0 K ground state.
- Valence orbital occupation attributes (total: 4): These attributes are the fraction-weighted average of the number of valence electrons in each of the *s*, *p*, *d*, and *f* orbitals divided by the fraction-weighted average of the total number of valence electrons.
- Ionic compound attributes (total: 3): The first attribute is Boolean (i.e., 0 or 1), indicating whether it is possible to form a neutral ionic compound assuming that each element is in one of its common charge states. For the other two attributes, the mean and maximum ionic character of a compound are computed using the electronegativities of the elements in the compound.

All of these attributes are continuous or integer numeric variables, except for the Boolean ionic compound attribute mentioned above.

The full dataset for analysis and model building consists of these 145 features and the raw data compiled from [1], i.e., 150 features for 9,590 compounds. Of these 150 features, the chemical formula will not be used as a predictor but instead as labels for the compounds. Further, the target/response variable is the heat of formation; thus, 148 features are available for modeling as explanatory variables.

4.2. Exploratory Data Analyses

All exploratory data analyses were carried out with R (v. 4.1.0); the R script for these analyses is presented in Appendix A. Of the 9,590 materials in the full dataset, 64 have one component, 2,743 have two components, and 6,783 have three components. Since many of the features in the dataset are artificially created from the chemical formula and elemental properties using Magpie, we will primarily take a closer look at the features in the original raw data compiled from the CMR. These features are the volume of the unit cell, the sum of the atomic masses, the number of atoms in the chemical formula, and the heat of formation (the first three features will be referred to as the volume, atomic mass, and number of atoms hereinafter).

Table 1. Summary statistics for the original features taken from the CMR.

	Unit Cell Volume [Å^3]	Atomic Mass [g/mol]	Number of Atoms	Heat of Form. [eV/atom]
Minimum	10.7	2.0	1	-4.123
1st Qu.	117.7	404.9	7	-1.183
Median	238.8	776.2	12	-0.655
Mean	318.0	1064.3	16	-0.894
3rd Qu.	423.8	1437.0	24	-0.371
Maximum	2270.2	7609.1	184	0.338

Table 1 presents the summary statistics for these four features. From these data, we see that all of these features except the heat of formation have asymmetric distributions with long tails at larger values. The heat of formation also has an asymmetric distribution with a long tail towards smaller values (larger negative values). In addition, there are some materials with both positive and negative heats of formation according to the statistics in Table 1. These observations are

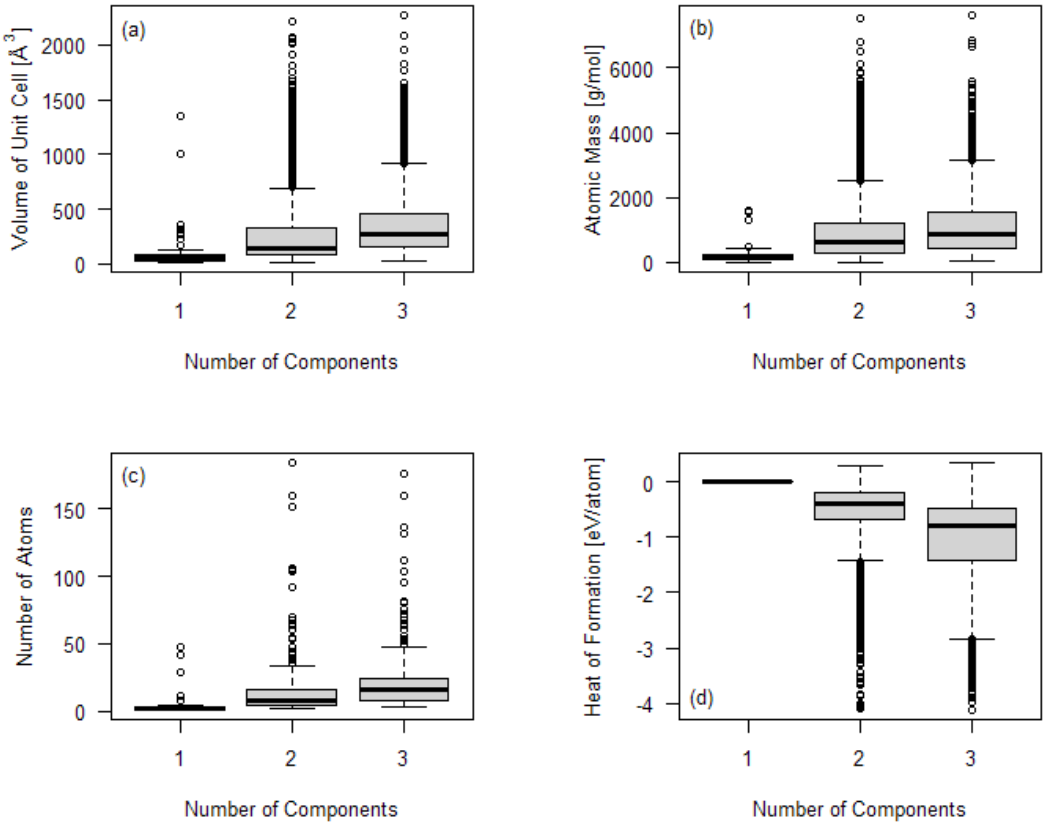

Figure 1. Boxplots of the (a) volume, (b) atomic mass, (c) number of atoms, and (d) heat of formation according to the number of components in the chemical formula.

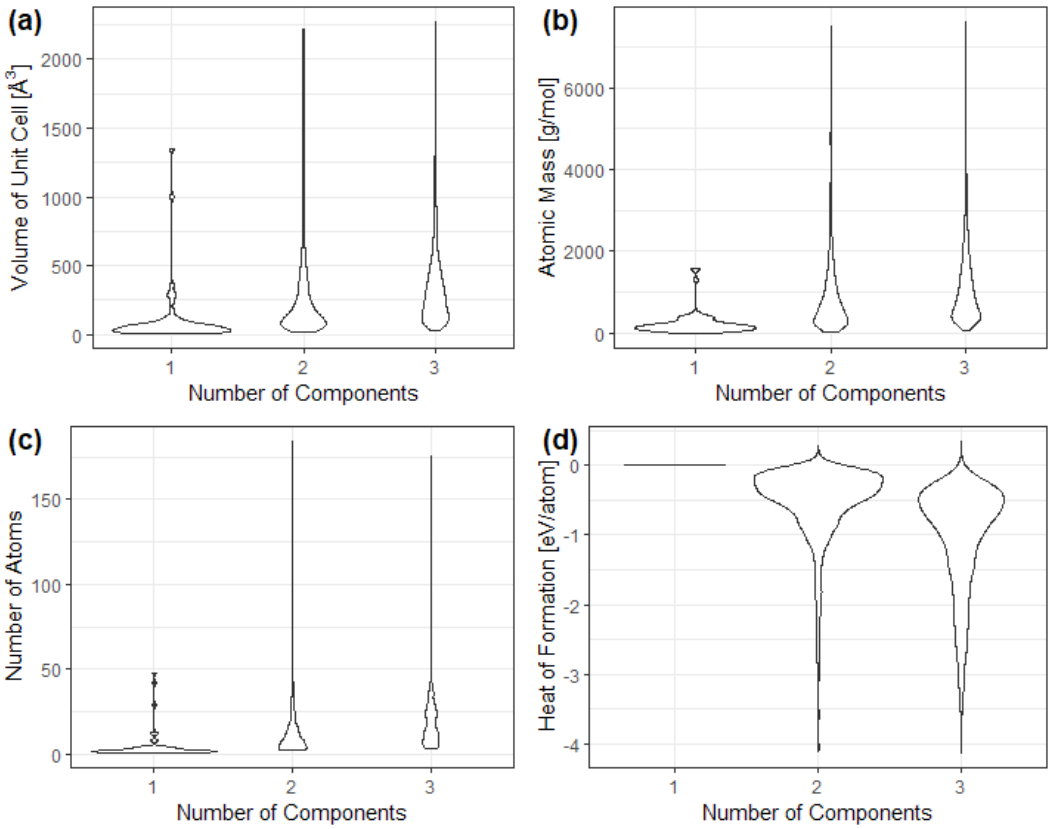

Figure 2. Violin plots of the (a) volume, (b) atomic mass, (c) number of atoms, and (d) heat of formation according to the number of components in the chemical formula.

confirmed from the boxplots in Fig. 1, where the boxplots are broken down according to the number of components. For all four features, we see that the tails of the distributions are longer for materials with two and three components. Notably, the boxplot for the heat of formation for one-component materials is simply a line at zero. This is because the heat of formation is zero for elemental materials, as stated in Section 1. Finally, the correlations ρ between the volume, atomic mass, and number of atoms and the heat of formation are rather low: 0.005, 0.122, and -0.160, respectively. However, the correlations between the volume, atomic mass, and number of atoms are all greater than 0.618, with the largest correlation ($\rho = 0.779$) between the volume and the atomic mass. These large correlations are not surprising, as these features are related by well-known formulas and materials constants, e.g., the volume and mass are related by the density.

Figure 2 shows violin plots of the volume, atomic mass, number of atoms, and heat of formation. These plots provide a better picture of how skewed the distributions are depending on the number of components. For materials with one component, the distributions are highly skewed for all features. However, the distributions are not as skewed for materials with two and three components, although there is still a high amount of skew for the heat of formation for materials with two components.

We now take a closer look at how the heat of formation varies according to the volume, atomic mass, and number of atoms using the scatterplots in Fig. 3. Since the heat of formation is zero for one-component materials, all of the data for these materials would lie on the *x* axis; hence, these data were omitted from the scatterplots, and the data were separately plotted for two- and three-

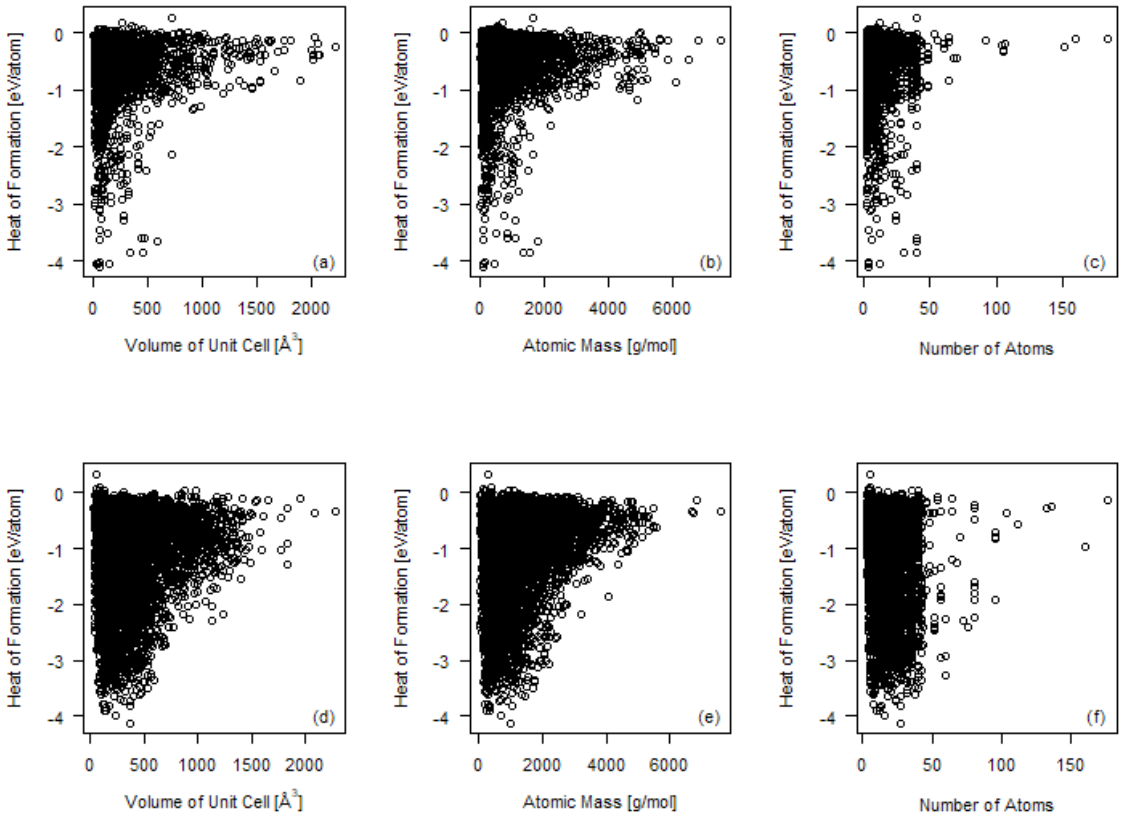

Figure 3. Scatterplots of the heat of formation versus the (a) volume, (b) atomic mass, and (c) number of atoms for two-component materials and (d) volume, (e) atomic mass, and (f) number of atoms for three-component materials.

component materials. For the two-component materials in Figs. 3(a)–(c), we see that most of the data is concentrated in the upper-left corner. According to Fig. 3(c), most two-component materials have no more than ~50 atoms; those that do have more than 50 atoms have a heat of formation that lies between –1 and 0 eV/atom. We find similar results for the three-component materials in Figs. 3(d)–(f). However, there appears to be a wider range of heats of formation compared to the two-component materials, which are mainly concentrated between –1 and 0 eV/atom. Moreover, some three-component materials with >50 atoms have heats of formation that are less than –1 eV/atom.

In addition, we looked at the summary statistics for the Magpie features and the correlations between the Magpie features and the heat of formation. Most of the features were not strongly correlated (i.e., $|\rho| < 0.5$) with the heat of formation. Some of these features did have moderate to large correlations ($0.5 < \rho < 0.789$). These include multiple features related to the covalent radius, electronegativity, number of *p* valence electrons, and space group number. In addition, the fractions of valence electrons in the *p* and *d* orbitals and the mean and maximum ionic character were also significantly correlated ($\rho > 0.5$). Notably, six features related to the number of unfilled *f* states did not have correlations with the heat of formation. After taking a closer look at the data, this is because all of the materials in the dataset had values of zero for these six attributes. This

makes sense since none of the materials in the dataset appear to contain elements with atoms having electrons in their *f* orbitals (i.e., lanthanides and actinides).

The results of the exploratory data analyses allow us to make two modifications to the dataset. Since one-component materials have a heat of formation of zero, we will predict that any new unseen material with only one component has zero heat of formation with no error. Thus, we will exclude materials with only one component from the ML models built in this work. Further, we can exclude the six Magpie features related to the number of unfilled *f* states since they have the same value of zero for all materials. Thus, for the ML models discussed later, the dataset will contain 142 features to predict the heats of formation of 9,526 two- and three-component materials.

5. Methods

All analyses and modeling were carried out using R (v. 4.1.0) on a PC equipped with an AMD Ryzen 3800X CPU (8 cores/16 threads, 3.9/4.5 GHz base/boost clocks) and 32 GB of RAM. All random number seeds were generated by selecting an arbitrary number of digits from a random number generated with a uniform distribution in the interval [0, 1]. These seeds were not changed to improve model performance. Appendix B presents the R script for the entire modeling process.

5.1. Training/Test Sets and Cross-Validation

After processing the dataset to remove the six predictors related to the number of unfilled *f* states and all 64 one-component materials, the caret (v. 6.0-88) package was used to create training and test sets. Given the size of the dataset, we used a 70%–30% split for the training and test sets, respectively. The caret package uses stratified sampling to ensure that there is a roughly equal representation of all values of the response variable. In addition, the training and test sets contained roughly the same proportions of two- and three-component materials. Since our dataset is of moderate size (just over 9,500 materials), we used repeated 10-fold CV with five repeats according to best practices [22]. To ensure that the same CV folds were used for each ML technique, the same random number seed was set prior to building each model.

5.2. Machine Learning Techniques

Instead of manually tuning a linear regression model from the available predictors, three model selection techniques were used to build regression models; these techniques are forward selection, backward elimination, and stepwise selection. Given the relatively large number of predictors, best subset selection is not possible without a large amount of computational resources. The forward selection process begins with a model with no predictors and then adds predictors one-by-one until there is no further improvement in model performance using some criterion such as Mallow's C_p or Akaike's information criterion (AIC). In contrast, backward selection starts with a model containing all predictors and removes predictors until there is no improvement in model performance. Finally, stepwise selection combines forward selection and backward elimination in that predictors are added and removed from the model at any stage of the model-building process.

In R, the leaps (v. 3.1) package was used to implement these techniques with caret. This package has one tuning parameter that limits the maximum number of predictors in the model. For forward selection and backward elimination, this parameter was set to 100 predictors. This parameter could have been set higher but was limited owing to the very small increase in model performance (discussed later in Section 5.4). For stepwise selection, this parameter was set to 20 predictors since any higher value required an excessively long computation time.

In addition, we employed two methods that apply regularization to linear regression: ridge regression and the lasso. In brief, these methods add a "penalty" term to the least-squares minimization process, thereby shrinking the estimates of the regression coefficients. The difference between these methods is the formulation of the penalty term, which uses l_1 and l_2 norms in the lasso and ridge regression, respectively [23]. The overall benefit of these methods is the reduction in variance compared to linear regression at the cost of a slight increase in bias. Moreover, the l_1 norm in the lasso allows it to perform variable selection since it may reduce some coefficient estimates to zero. In R, the glmnet (v. 4.1-2) package was utilized; the penalty term λ was set to vary from 10^{-6} to 10^{3} during training for both ridge regression and the lasso.

Finally, three tree-based methods were selected: the decision tree, random forests, and boosting. The decision tree algorithm grows a single tree using successive binary recursive splits of the entire predictor space with a greedy approach. For regression trees, a tree is typically grown to minimize the residual sum of squares (RSS) and stops once the RSS can no longer be improved. A deep and complex tree could produce a model that provides good predictions on the training set but poor performance on the test set, a phenomenon known as overfitting. This overfitting issue can be overcome with tree pruning techniques and other parameters, e.g., the minimum number of observations in a node. Further, decision trees suffer from high variance since only one tree is grown.

Other tree-based approaches have been developed to mitigate these issues. Random forests and boosting both build multiple decision trees, i.e., an ensemble, but with some algorithmic differences. In random forests, a random number of predictors is considered at each split in a decision tree instead of all possible predictors (as done in the decision tree); this has the effect of decorrelating the trees since there will be a wide variety of trees that have different structures. In boosting, trees are built sequentially, and each subsequent tree "learns" from the mistakes of the previously built tree.

In R, the rpart (v. 4.1-15), ranger (v. 0.12.1), and gbm (v. 2.1.8) packages were used for the decision tree, random forests, and boosting methods. Since caret does not allow the number of trees to be tuned for random forests, it was set to 1,500, i.e., roughly 10 times the total number of predictors (as suggested by [24]) to ensure a sufficient number of trees in the ensemble. The number of predictors considered at each split (10–100 in steps of 10) and the minimum node size (5 and 10) were tuned. For boosting, we fixed the number of trees in the ensemble to 1,500 and tuned the interaction depth (3, 5, and 7), which controls the depth of the tree, and the learning rate (0.1, 0.5, and 1). It is noted that, in general, the number of trees should be tuned for boosting; however, given the setup for CV and the computation times needed for boosting (more than 6 h for the hyperparameters mentioned above with repeated 10-fold CV), it was necessary to fix its value to ensure a reasonable computation time and allow for adequate troubleshooting.

5.3. Performance Metrics

We used the root mean squared error (RMSE) and MAE to select the optimal hyperparameters for each ML model on the cross-validated training set and to assess the performance of each model on the test set. The RMSE and MAE are calculated as follows:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{n}}, \tag{1}$$

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|x_i - \hat{x}_i|}{n}, \tag{2}$$

Table 2. Performance metrics for the ML models on the cross-validated training set and test set. All values are reported in [meV/atom], and the training-set performance metrics are reported as mean ± one SD.

	Training Set		Test Set	
Method	RMSE	MAE	RMSE	MAE
Forward Selection	310 ± 11	235 ± 8	313	233
Backward Elimination	308 ± 10	234 ± 7	314	232
Stepwise Selection	346 ± 31	261 ±24	350	260
Ridge Regression	296 ± 10	222 ± 7	299	220
Lasso	277 ± 10	210 ± 6	278	208
Decision Tree	460 ± 29	344 ± 20	478	357
Random Forests	132 ± 8	91 ± 4	138	92
Boosting	98 ± 6	69 ± 3	97	68

where x_i is the actual value of the heat of formation of the ith material, $\hat{x}_i$ is the value predicted for the heat of formation of the ith material by an ML model (on either the training or test set), and n is the total number of materials in either the training or test set.

5.4. Hyperparameter Selection Using the Training Set

In general, the hyperparameters that produce the lowest RMSE and MAE on the cross-validated training set should be selected for use in the final model whose performance will be tested on the test set. This guideline was applied to ridge regression, the lasso, random forests, and boosting. For the decision tree, there were no hyperparameters to tune, and the fully grown tree (without pruning) was used as the final model. For ridge regression and the lasso, the best performance on the training set was obtained when $\lambda = 10^{-2}$ and 10^{-5}, respectively. For random forests, the hyperparameters of the final model are number of trees = 1,500, number of predictors = 30, and minimum node size = 5. For boosting, the hyperparameters of the final model are number of trees = 1,500, interaction depth = 7, and learning rate = 0.1.

For forward selection, backward elimination, and stepwise selection, the coefficient of determination R^2 was also used to help select the best model. For these models, the number of predictors was tuned. The best hyperparameter value was selected to be the one that provided a relatively low RMSE and MAE (and a relatively high R^2) while balancing the number of predictors. That is, we tried to avoid adding too many predictors to the model to avoid overfitting and a minimal improvement in performance (as a visual aid, the RMSE, MAE, and R^2 are plotted versus the number of predictors in Figs. C.1 and C.2 in Appendix C). As a result, the final models for forward selection and backward elimination used 50 predictors. The stepwise selection model used 20 predictors, although this choice was limited owing to the computation time.

6. Results and Discussion

6.1. Training- and Test-Set Results

The values of the RMSE and MAE for all ML models on the training and test sets are summarized in Table 2. The results in Table 2 for the training set are reported as mean ± one standard deviation (SD) owing to the use of repeated 10-fold CV. On both the training and test sets, the decision tree has the worst performance with high values for the RMSE and MAE.

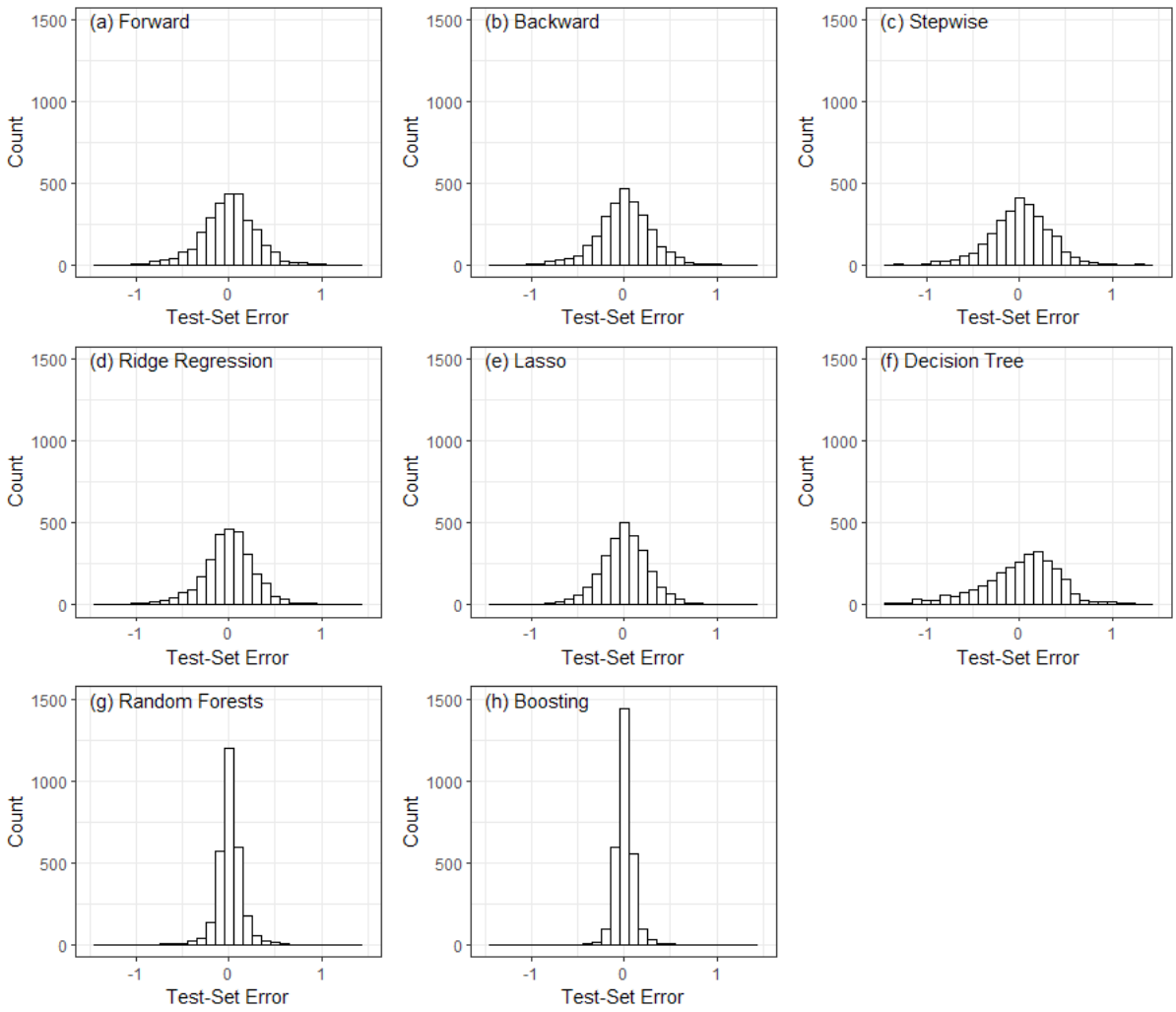

Figure 4. Histograms of the test-set errors for (a) forward selection, (b) backward elimination, (c) stepwise selection, (d) ridge regression, (e) the lasso, (f) decision tree, (g) random forests, and (h) boosting.

However, the ensemble tree-based methods provide the best performance on both the training and test sets, with random forests and boosting achieving MAEs of 92 and 68 meV/atom on the test set. The regression-based techniques provide performance between these two extremes, with RMSEs and MAEs closer to those for the decision tree. Forward selection and backward elimination achieve similar performance, with stepwise selection providing slightly worse performance due to the lower number of predictors. Ridge regression and the lasso both provide notable improvements in performance compared to the other regression-based techniques, but their RMSEs and MAEs are approximately double those of random forests on the training and test sets or higher. Notably, the use of repeated 10-fold CV provides very good estimates of the test-set performance because all values of the RMSE and MAE on the test set fall within one SD of the mean values for the RMSE and MAE on the training set.

Figure 4 shows histograms of the prediction errors $(x_i - \hat{x}_i)$ on the test set for each of the ML models. Most of these histograms appear to be bell-shaped, except for the decision tree, which has noticeable skew. The distributions for the regression-based models are quite similar and rather

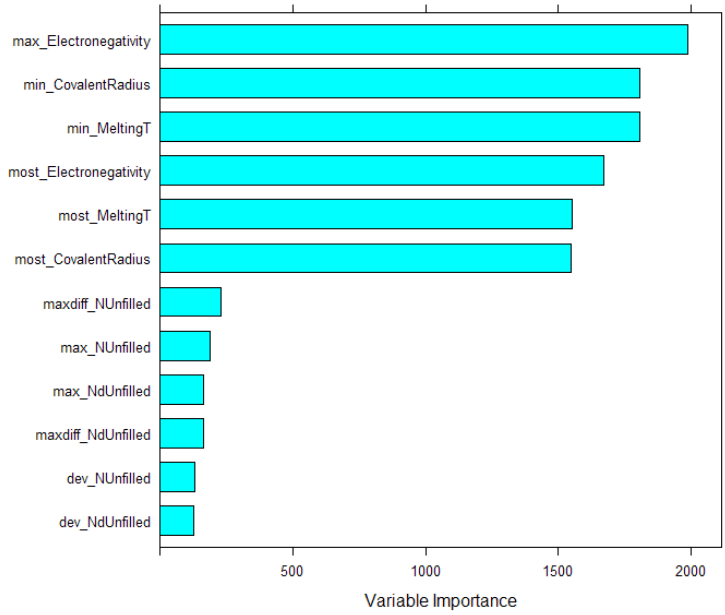

Figure 5. Importance of all of the features used in the construction of the decision tree.

broad over the range from –1.5 to 1.5. In contrast, random forests and boosting have less spread in their distributions, as the test-set errors fall within a much narrower range. Moreover, the errors are overwhelmingly clustered and centered around zero. These results indicate that random forests and boosting are much better models for predicting the heat of formation since the resulting error will most likely be smaller than those for the regression methods.

6.2. Features Used in the ML Models

We now take a look at the features used in the ML models. In forward selection, 12 features related to the numbers of valence electrons (in the *s*, *p*, and *d* shells and in total) and 10 features related to the numbers of unfilled states (in the *s*, *p*, and *d* shells and in total) were the most numerous, followed by features related to the magnetic moment (4). Similarly, backward elimination preferred features related to the numbers of valence electrons (6) and the numbers of unfilled states (4) but included all of the stoichiometric features. In addition, four features related to the electronegativity were used. Finally, stepwise selection included features related to the numbers of valence electrons (3), the numbers of unfilled states (3), and the electronegativity (3).

By its nature, ridge regression does not perform variable selection; hence, all 142 predictors are present in the model. In contrast, the lasso can perform variable selection. From the coefficients for the final lasso model, it appears that 20 features were removed from the model (i.e., their coefficients were zero), with five related to the numbers of valence electrons and six related to the numbers of unfilled states.

Figure 5 shows the importance of all of the features used in the construction of the decision tree. Half of the 12 features used in its construction are related to the numbers of unfilled states (although they have the lowest importance) and two are related to the melting temperature. For random forests and boosting, we look at the top 25 most important features since multiple trees are grown in these algorithms. In Fig. 6(a), four of the top six most important variables are related to the electronegativity and two of the top six are related to the ionic character for random forests.

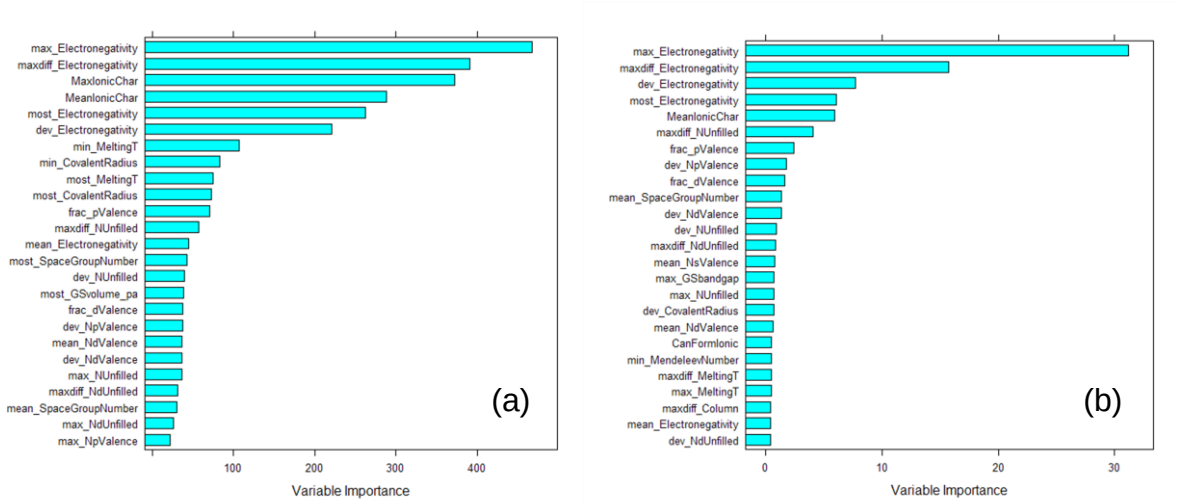

Figure 6. Variable importance plots for (a) random forests and (b) boosting.

Table 3. Comparison of the MAEs of notable models in the literature and the MAEs of random forests and boosting in our work.

Technique	MAE [meV/atom]
Linear Regression (Stepwise) [12]	80
REPTree [8]	88.2
Random Forests [15]	80
ElemNet (DNN) [16]	50
CGCNN [17]	39
SchNet (DTDN) [18]	35
MEGNet (Graph Network) [19]	28
This Work: Random Forests	92
This Work: Boosting	68

For boosting in Fig. 6(b), the top four most important variables are related to the electronegativity, followed by one variable related to the ionic character. For these tree-based algorithms, we note that some of the most important features are also those that had high correlations with the heat of formation, as discussed in Section 4.2, e.g., the features related to the electronegativity and ionic character. The forward selection, backward elimination, and stepwise selection methods also utilize features related to the electronegativity (two, four, and three features, respectively), but only backward elimination uses one feature related to the ionic character. Finally, all variables related to electronegativity and ionic character are retained in the lasso model.

6.3. Comparison with Literature Results

As noted in Section 3, the MAE of 96 meV/atom is derived from DFT calculations and actual experimental data [13]; thus, it does not represent the performance of an actual ML model. However, it is a useful benchmark, as an ML model that achieves an MAE that is less than 96 meV/atom would perform better than DFT calculations. Of the models utilized in this work, two perform better than this benchmark: random forests and boosting.

Table 3 summarizes the results from the literature mentioned in Section 3 along with the results for our two models that exceed the benchmark from [13]. We find that our models compare quite favorably to those in the literature, especially since we did not construct complicated neural network models, utilize graph networks, use additional complex features, or perform extensive hyperparameter tuning. The MAE of 92 meV/atom for random forests in our work is higher than the other random forests model employed in [15]. However, we did not tune the number of trees in our model. Further, our boosting model achieves an MAE of 68 meV/atom, which is better than the linear regression model in [12], REPTree in [8], and the random forests model in [15]. Moreover, the boosting model was trained over a very limited hyperparameter space; hence, it is likely that its performance could be improved.

7. Limitations and Plans for Future Work

There are many potential limitations for the work done in this study. As discussed earlier, the limitations on time and computational resources constrained our tuning of the hyperparameters of the ML models. This is particularly notable for stepwise selection, which was limited to 20 predictors, and it is expected that additional predictors would improve model performance. The addition of a 20th predictor to the stepwise selection model lowered the training-set MAE by 14 meV/atom, and the forward selection and backward elimination models used 50 predictors before the improvement in performance was minimal. However, it is unlikely that additional predictors will decrease the RMSE and MAE for stepwise selection to less than the benchmark of 96 meV/atom. As mentioned earlier, the number of trees was not tuned but is typically tuned for boosting. According to [25], boosting often requires many trees owing to the sequential nature of learning, which allows it to continue to improve upon its mistakes. Hence, it is likely that an increase in the number of trees will improve the performance for boosting. It is also likely that further performance improvements can be realized for both random forests and boosting with further hyperparameter tuning, as both of these techniques were tuned within a rather limited and coarse hyperparameter space.

In addition, our regression models only considered linear terms; that is, none of the predictors were transformed. It is possible that further improvements in these models may be achieved with appropriate transformations. However, this would require a tedious process of testing many potential transformations, requiring considerable computational resources. It may also be worthwhile to consider using principal components analysis (PCA) to reduce the number of predictors before building regression models. A small number of principal components (<10) may allow the regression models to be tuned manually and more easily avoid overfitting.

Most importantly, our model is only valid for materials containing 1–3 components and is not expected to perform well for materials containing four or more components. Although only a few models explicitly appear to use the feature related to the number of components (notably, backward elimination), some of the Magpie-generated features could be significantly altered with the addition of more components, e.g., those related to the atomic weight would most likely increase owing to the additional components. Moreover, the volume, mass, and number of atoms could all significantly change, perhaps beyond their ranges of values in our dataset.

In future work, the hyperparameters of the random forests and boosting models should be tuned to improve performance. Moreover, we may consider using additional features, particularly those discussed in [15]. As indicated by the results in Table 3, ML models based on neural networks can further improve performance, and we may consider using other boosting algorithms such as XGBoost, which has better performance and provides better scalability [26]. Finally, it

may be useful to investigate how the heat of formation varies according to the particular elements in a material. Thus, further study of the clustering behavior related to elements that are more commonly present in materials, e.g., O, N, Al, etc., may provide further insights and allow additional refinements to the modeling process.

8. Conclusions

We have developed ML models that predict the heat of formation of one-, two-, and three-component materials. Our best model uses the boosting technique and achieves a low RMSE and MAE of 97 and 68 meV/atom on the test set. The MAE of this model compares quite favorably to many models in the literature, even with limited parameter tuning. Moreover, it exceeds the benchmark value of 96 meV/atom derived from DFT calculations and experimental values. Our final modeling process predicts the heat of formation to be 0 meV/atom for any one-component material and uses the prediction from the boosting model for two- and three-component materials. Given the accuracy of the boosting model, it would be suitable for many MI tasks such as materials selection to provide accurate predictions of the heat of formation.

References

1. A. Agrawal and A. Choudhary, "Perspective: Materials informatics and big data: Realization of the "fourth paradigm" of science in materials science," *APL Mater.*, vol. 4, Art. no. 053208, 2016.
2. J. Hill, G. Mulholland, K. Persson, R. Seshadri, C. Wolverton, and B. Meredig, "Materials science with large-scale data and informatics: Unlocking new opportunities," *MRS Bull.*, vol. 41, pp. 399–409, May 2016.
3. A. Agrawal and A. Choudhary, "Deep materials informatics: Applications of deep learning in materials science," *MRS Commun.*, vol. 9, pp. 779–792, 2019.
4. A. Jain *et al.*, "Commentary: The Materials Project: A materials genome approach to accelerating materials innovation," *APL Mater.*, vol. 1, Art. no. 011002, 2013.
5. L. Himanen, A. Geurts, A. S. Foster, and P. Rinke, "Data-driven materials science: Status, challenges, and perspectives," *Adv. Sci.*, vol. 6, Art. no. 1900808, 2019.
6. A. Y.-T. Wang *et al.*, "Machine learning for materials scientists: An introductory guide toward best practices," *Chem. Mater.*, vol. 32, pp. 4954–4965, 2020.
7. N. Wagner and J. M. Rondinelli, "Theory-guided machine learning in materials science," *Front. Mater.*, vol. 3, Art. no. 28. Jun. 2016.
8. L. Ward, A. Agrawal, A. Choudhary, and C. Wolverton, "A general-purpose machine learning framework for predicting properties of inorganic materials," *NPJ Comput. Mater.*, vol. 2, Art. no. 16028, 2016.
9. V. Stanev *et al.*, "Machine learning modeling of superconducting critical temperature," *NPJ Comput. Mater.*, vol. 4, Art. no. 29, 2018.
10. O. Isayev, C. Oses, C. Toher, E. Gossett, S. Curtarolo, and A. Tropsha, "Universal fragment descriptors for predicting properties of inorganic crystals," *Nat. Commun.*, vol. 8, Art. no. 15679, 2017.
11. B. Meredig *et al.*, "Combinatorial screening for new materials in unconstrained composition space with machine learning," *Phys. Rev. B*, vol. 89, 094104, 2014.
12. A. M. Deml, R. O'Hayre, C. Wolverton, and V. Stevanović, "Predicting density functional theory total energies and enthalpies of formation of metal-nonmetal compounds by linear regression," *Phys. Rev. B*, vol. 93, Art. no. 085142, 2016.

13 S. Kirklin, *et al.*, "The Open Quantum Materials Database (OQMD): Assessing the accuracy of DFT formation energies," *NPJ Comput. Mater.*, vol. 1, Art. no. 15010, Dec. 2015.
14. J. E. Saal, S. Kirklin, M. Aykol, B. Meredig, and C. Wolverton, "Materials design and discovery with high-throughput density functional theory: The Open Quantum Materials Database (OQMD)," *JOM*, vol. 65, pp. 1501–1509, Sept. 2013.
15. L. Ward *et al.*, "Including crystal structure attributes in machine learning models of formation energies via Voronoi tessellations," *Phys. Rev. B*, vol. 96, Art. no. 024104, 2017.
16. D. Jha *et al.*, "*ElemNet*: Deep learning the chemistry of materials from only elemental composition," *Sci. Rep.*, vol. 8, Art. no. 17593, 2018.
17. T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties," *Phys. Rev. Lett.*, vol. 120, Art. no. 145301, 2018.
18. K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller, "SchNet - A deep learning architecture for molecules and materials," *J. Chem. Phys.*, vol. 148, Art. no. 241722, 2018.
19. C. Chen, W. Ye, Y. Zuo, C. Zheng, and S. P. Ong, "Graph networks as a universal machine learning framework for molecules and crystals," *Chem. Mater.*, vol. 31, pp. 3564–3572, 2019.
20. Computational Materials Repository, 2019. "One, Two and Three Component References from OQMD," [Online]. Available: https://cmr.fysik.dtu.dk/oqmd123/oqmd123.html.
21. Wolverton Research Group. *Magpie*. Accessed: January 9, 2021. [Online]. Available: https://bitbucket.org/wolverton/magpie/.
22. B. Boehmke and B. Greenwell, *Hands-On Machine Learning with R*, Boca Raton, FL, USA: CRC Press, 2020, Sec. 2.4.1.
23. G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, New York, NY, USA: Springer, 2013, Sec. 6.2.
24. B. Boehmke and B. Greenwell, *Hands-On Machine Learning with R*, Boca Raton, FL, USA: CRC Press, 2020, Sec. 11.4.1.
25. B. Boehmke and B. Greenwell, *Hands-On Machine Learning with R*, Boca Raton, FL, USA: CRC Press, 2020, Sec. 12.3.
26. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," arXiv: 1603.02754.

Appendices

A. R Script for the Exploratory Data Analyses

```
# MATH 688: Data Analytics Capstone II
# Capstone Project: Exploratory Data Analyses
# Author: Isaiah Steinke
# Last Modified: June 17, 2021
# Written, Debugged, and Tested in R v. 4.1.0

# Load required libraries
# -----------------------
library(ggplot2) # v. 3.3.4
library(ggpubr)  # v. 0.4.0

# Import dataset
# --------------
heat <- read.csv("Full_Dataset.csv", header = TRUE)
```

```
# Look at the number of components
# --------------------------------
hist(heat$NComp, main = NULL, xlab = "Number of Components")
sum(heat$NComp == 1) # 64
sum(heat$NComp == 2) # 2743
sum(heat$NComp == 3) # 6783

# Summary statistics & plots: original data excluding Magpie features
# --------------------------------------------------------------------
summary(subset(heat, select = c(Volume, Mass, NoAtoms, HeatForm)))
cor(subset(heat, select = c(Volume, Mass, NoAtoms, HeatForm)))
pairs(subset(heat, select = c(Volume, Mass, NoAtoms, HeatForm)))

# Notably, none of the original predictors have significant
# correlations with HeatForm (the largest is 0.16 in magnitude).

# Violin plots
v1 <- ggplot(heat, aes(factor(NComp), Volume)) + geom_violin() +
      xlab("Number of Components") +
      ylab(expression("Volume of Unit Cell [Å"^{~3}*"]")) +
      theme_bw()
v2 <- ggplot(heat, aes(factor(NComp), Mass)) + geom_violin() +
      xlab("Number of Components") +
      ylab("Atomic Mass [g/mol]") +
      theme_bw()
v3 <- ggplot(heat, aes(factor(NComp), NoAtoms)) + geom_violin() +
      xlab("Number of Components") +
      ylab("Number of Atoms") +
      theme_bw()
v4 <- ggplot(heat, aes(factor(NComp), HeatForm)) + geom_violin() +
      xlab("Number of Components") +
      ylab("Heat of Formation [eV/atom]") +
      theme_bw()
ggarrange(v1, v2, v3, v4, ncol = 2, nrow = 2,
          labels = c("(a)", "(b)", "(c)", "(d)"))

# Scatterplots
ggplot(heat, aes(Volume, Mass, color = factor(NComp))) +
  geom_point(shape = 1) + theme_bw()
ggplot(heat, aes(Volume, NoAtoms, color = factor(NComp))) +
  geom_point(shape = 1) + theme_bw()
ggplot(heat, aes(Volume, HeatForm, color = factor(NComp))) +
  geom_point(shape = 1) + theme_bw()
ggplot(heat, aes(Mass, NoAtoms, color = factor(NComp))) +
  geom_point(shape = 1) + theme_bw()
ggplot(heat, aes(Mass, HeatForm, color = factor(NComp))) +
  geom_point(shape = 1) + theme_bw()
ggplot(heat, aes(NoAtoms, HeatForm, color = factor(NComp))) +
  geom_point(shape = 1) + theme_bw()

# It's difficult to distinguish points in these scatterplots since
# there are a lot of data points. Even using "alpha" argument (instead
# of "shape") doesn't improve things much.

# Set global plotting parameters
par(las = 1, mar = c(5.1, 6.1, 2.1, 2.1))
```

```
# Boxplots
par(mfrow = c(2, 2))
boxplot(Volume ~ factor(NComp), data = heat,
        xlab = "Number of Components",
        ylab = expression("Volume of Unit Cell [Å"^{~~3}*"]"))
text(0.6, 2150, "(a)")
boxplot(Mass ~ factor(NComp), data = heat,
        xlab = "Number of Components",
        ylab = "Atomic Mass [g/mol]")
text(0.6, 7250, "(b)")
boxplot(NoAtoms ~ factor(NComp), data = heat,
        xlab = "Number of Components",
        ylab = "Number of Atoms")
text(0.6, 175, "(c)")
boxplot(HeatForm ~ factor(NComp), data = heat,
        xlab = "Number of Components",
        ylab = "Heat of Formation [eV/atom]")
text(0.6, -3.9, "(d)")
par(mfrow = c(1, 1))

# Scatterplots; separate by number of components since the number of
# data points to plot is high
plot(heat$Mass[heat$NComp == 1], heat$Volume[heat$NComp == 1],
     main = NULL)
plot(heat$Mass[heat$NComp == 2], heat$Volume[heat$NComp == 2],
     main = NULL)
plot(heat$Mass[heat$NComp == 3], heat$Volume[heat$NComp == 3],
     main = NULL)

# It's nice to see the how the data separate by NComp. However, this
# would result in a large number of plots (6 × 3 = 18), and there
# doesn't appear to be any deep insights.

# Reset plot margins
par(mar = c(5.1, 4.1, 4.1, 2.1))

# Instead, let's just look at HeatForm vs. Mass, Volume, and NoAtoms
# for two and three components.
par(mfrow = c(2, 3))
plot(heat$Volume[heat$NComp == 2], heat$HeatForm[heat$NComp == 2],
     main = NULL, xlab = expression("Volume of Unit Cell [Å"^{3}*"]"),
     ylab = "Heat of Formation [eV/atom]")
text(2125, -4, "(a)")
plot(heat$Mass[heat$NComp == 2], heat$HeatForm[heat$NComp == 2],
     main = NULL, xlab = expression("Atomic Mass [g/mol]"),
     ylab = "Heat of Formation [eV/atom]")
text(7200, -4, "(b)")
plot(heat$NoAtoms[heat$NComp == 2], heat$HeatForm[heat$NComp == 2],
     main = NULL, xlab = expression("Number of Atoms"),
     ylab = "Heat of Formation [eV/atom]")
text(175, -4, "(c)")
plot(heat$Volume[heat$NComp == 3], heat$HeatForm[heat$NComp == 3],
     main = NULL, xlab = expression("Volume of Unit Cell [Å"^{3}*"]"),
     ylab = "Heat of Formation [eV/atom]")
text(2175, -4, "(d)")
plot(heat$Mass[heat$NComp == 3], heat$HeatForm[heat$NComp == 3],
     main = NULL, xlab = expression("Atomic Mass [g/mol]"),
```

```
     ylab = "Heat of Formation [eV/atom]")
text(7250, -4, "(e)")
plot(heat$NoAtoms[heat$NComp == 3], heat$HeatForm[heat$NComp == 3],
     main = NULL, xlab = expression("Number of Atoms"),
     ylab = "Heat of Formation [eV/atom]")
text(168, -4, "(f)")
par(mfrow = c(1, 1))

# Reset plot area
par(las = 0)

# Summary statistics: all features
# --------------------------------
summary(heat[, 2:150])
cor(heat[, 2:149], heat$HeatForm)

# Most features have correlations with HeatForm that are <0.5.
# Some of the features related to CovalentRadius, Electronegativity,
# NpValence, SpaceGroupNumber, frac_pValence, frac_dValence, MaxIonicChar,
# and MeanIonicChar have correlations with HeatForm that are >0.5. The
# largest is for MeanIonicChar (-0.788977848). In addition, there are
# features that have NA for the correlations; this is because these
# features have the same value of zero for every compound. These features
# are the six features related to NfUnfilled. Thus, these features should
# be removed before building models since they will be useless for
# prediction.
```

B. R Script for Model Building and Performance Assessment

```
# MATH 688: Data Analytics Capstone II
# Capstone Project: Model Building and Assessment
# Author: Isaiah Steinke
# Last Modified: July 2, 2021
# Written, Debugged, and Tested in R v. 4.1.0

# ======================================================================
# Load required libraries
# ======================================================================
library(ggplot2) # v. 3.3.5
library(ggpubr)  # v. 0.4.0
library(caret)   # v. 6.0-88
library(leaps)   # v. 3.1
library(glmnet)  # v. 4.1-2
library(rpart)   # v. 4.1-15
library(ranger)  # v. 0.12.1
library(gbm)     # v. 2.1.8

# ======================================================================
# Import dataset
# ======================================================================
heat <- read.csv("Full_Dataset.csv", header = TRUE)

# ======================================================================
# Process dataset
# ======================================================================
# From our exploratory data analyses, we found that six features
```

```
# had the same value of zero for all compounds. Thus, these features
# can be removed since they will have no explanatory use in the
# models. In addition, we will remove all one-component compounds
# since the heat of formation is zero for these compounds. Thus,
# our final model process will first check for the number of
# components before applying the ML model to two- and three-component
# materials.

# Remove useless features
del.features <- c("mean_NfUnfilled", "maxdiff_NfUnfilled",
                  "dev_NfUnfilled", "max_NfUnfilled",
                  "min_NfUnfilled", "most_NfUnfilled")
heat <- within(heat, rm(list = del.features))

# Alternative method
del.features.indexes <- which(names(heat) %in% del.features)
heat <- heat[, -del.features.indexes]

# Remove one-component materials
del.onecomp <- which(heat$NComp == 1)
heat <- heat[-del.onecomp, ]

# Clean up environment
rm(del.features, del.onecomp)

# Create training and test sets
# -----------------------------
# Since we have a good amount of data, we'll use a 70/30 split
# for the training/test sets.
set.seed(26448)
indexes <- createDataPartition(heat$HeatForm, times = 1,
                               p = 0.7, list = FALSE)
heat.train <- heat[indexes, ]
heat.test <- heat[-indexes, ]

# The first column, "Formula," is simply the name of the compound
# and only useful as a label. To make things easier in the
# following analyses, we'll store the formulas in separate vectors
# for the training/test sets and just have the predictors and
# response variables in our dataset.

heat.train.labels <- heat.train$Formula
heat.train <- heat.train[, -1]
heat.test.labels <- heat.test$Formula
heat.test <- heat.test[, -1]

# Create a model matrix for the test data that will be useful
# when computing predictions
testdata <- model.matrix(HeatForm ~ ., heat.test)

# Clean up environment
rm(indexes)

# ====================================================================
# Set the cross-validation parameters for caret
# ====================================================================
cv.params <- trainControl(method = "repeatedcv", number = 10,
```

```
                          repeats = 5)

# Note: In the following, we build models with the same random
# number seed to ensure that the cross-validation procedure is
# performed in the same way for each method.

# =====================================================================
# Regression models: Subset selection methods
# =====================================================================
# We'll be using regsubsets in caret via the train function.
# Since the output of regsubsets doesn't provide easy access to
# predictions, we'll have to do some more work with the output.

# Forward selection
# -----------------
# Rather fast, so we'll have it fit up to 100 variables.
set.seed(723)
fwd.model <- train(HeatForm ~ ., data = heat.train,
                   method = "leapForward", trControl = cv.params,
                   tuneGrid = data.frame(nvmax = 1:100))

# Analyze results; compute test-set metrics
fwd.model$results
write.csv(fwd.model$results, file = "fwd.csv", row.names = FALSE)
plot(fwd.model, metric = "RMSE")
plot(fwd.model, metric = "Rsquared")
plot(fwd.model, metric = "MAE")

# Select model with 50 predictors since models with more
# predictors suffer from diminishing returns.

coef.values <- coef(fwd.model$finalModel, 50)
coef.names <- names(coef.values)
fwd.preds <- testdata[, coef.names] %*% coef.values
fwd.err <- heat.test$HeatForm - fwd.preds
sqrt(mean(fwd.err^2)) # RMSE
mean(abs(fwd.err)) # MAE

# Backward selection
# ------------------
# Again, this is fairly fast with only one thread.
set.seed(723)
bwd.model <- train(HeatForm ~ ., data = heat.train,
                   method = "leapBackward", trControl = cv.params,
                   tuneGrid = data.frame(nvmax = 1:100))

# Analyze results; compute test-set metrics
bwd.model$results
write.csv(bwd.model$results, file = "bwd.csv", row.names = FALSE)
plot(bwd.model, metric = "RMSE")
plot(bwd.model, metric = "Rsquared")
plot(bwd.model, metric = "MAE")

# Select model with 50 predictors since models with more
# predictors suffer from diminishing returns.

coef.values <- coef(bwd.model$finalModel, 50)
```

```
coef.names <- names(coef.values)
bwd.preds <- testdata[, coef.names] %*% coef.values
bwd.err <- heat.test$HeatForm - bwd.preds
sqrt(mean(bwd.err^2)) # RMSE
mean(abs(bwd.err)) # MAE

# Stepwise selection
# ------------------
# We have to limit nvmax to 20; higher values require a long
# time to finish, even with multiple threads. (In fact, I had
# to force-close R after an hour with nvmax = 35.)
set.seed(723)
step.model <- train(HeatForm ~ ., data = heat.train,
                    method = "leapSeq", trControl = cv.params,
                    tuneGrid = data.frame(nvmax = 1:20))

# Analyze results; compute test-set metrics
step.model$results
write.csv(step.model$results, file = "step.csv", row.names = FALSE)
plot(step.model, metric = "RMSE")
plot(step.model, metric = "Rsquared")
plot(step.model, metric = "MAE")

# Select model with 20 predictors. It does appear that performance
# can be improved with more predictors, but it takes too long if
# I increase nvmax (even to 25).

coef.values <- coef(step.model$finalModel, 20)
coef.names <- names(coef.values)
step.preds <- testdata[, coef.names] %*% coef.values
step.err <- heat.test$HeatForm - step.preds
sqrt(mean(step.err^2)) # RMSE
mean(abs(step.err)) # MAE

# Clean up environment
rm(coef.values, coef.names)

# ====================================================================
# Regression models: Ridge regression & lasso
# ====================================================================
# Use glmnet; alpha = 0 for ridge regression and alpha = 1 for
# lasso. Tune lambda values.

# Ridge regression
# ----------------
# Set tuning parameters
rr.params <- expand.grid(alpha = 0,
                         lambda = 10^seq(3, -6, length = 10))

# Build model
set.seed(723)
rr.model <- train(HeatForm ~ ., data = heat.train,
                  method = "glmnet", trControl = cv.params,
                  tuneGrid = rr.params)

# Analyze results; compute test-set metrics
rr.model$results
```

```
write.csv(rr.model$results, file = "rr.csv", row.names = FALSE)

# Select lambda = 1e-02 as the best parameter value.

rr.preds <- predict(rr.model$finalModel, s = 1e-02,
                    newx = testdata[, -1])
rr.err <- heat.test$HeatForm - rr.preds
sqrt(mean(rr.err^2)) # RMSE
mean(abs(rr.err)) # MAE

# Lasso
# -----
# Set tuning parameters
lasso.params <- expand.grid(alpha = 1,
                            lambda = 10^seq(3, -6, length = 10))

# Build model
set.seed(723)
lasso.model <- train(HeatForm ~ ., data = heat.train,
                     method = "glmnet", trControl = cv.params,
                     tuneGrid = lasso.params)

# Analyze results; compute test-set metrics
lasso.model$results
write.csv(lasso.model$results, file = "lasso.csv", row.names = FALSE)

# Select lambda = 1e-05 as the best parameter value.

lasso.preds <- predict(lasso.model$finalModel, s = 1e-05,
                       newx = testdata[, -1])
lasso.err <- heat.test$HeatForm - lasso.preds
sqrt(mean(lasso.err^2)) # RMSE
mean(abs(lasso.err)) # MAE
lasso.coef <- predict(lasso.model$finalModel, type = "coefficients",
                      s = 1e-05)

# Clean up environment
rm(rr.params, lasso.params)

# ================================================================
# Tree-based methods
# ================================================================
# For the basic decision tree, we'll use rpart. For random forests,
# we'll use ranger since we can leverage the built-in parallel
# processing functionality. Finally, we'll use gbm for boosted
# decision trees.

# Decision tree
# -------------
set.seed(723)
tree.model <- train(HeatForm ~ ., data = heat.train,
                    method = "rpart", trControl = cv.params)

# Analyze results; compute test-set metrics
tree.model$results
tree.preds <- predict(tree.model$finalModel, newdata = heat.test)
tree.err <- heat.test$HeatForm - tree.preds
```

```
sqrt(mean(tree.err^2)) # RMSE
mean(abs(tree.err)) # MAE

# Variable importance
tree.vi <- tree.model$finalModel[[12]]
barchart(tree.vi[12:1], xlab = "Variable Importance")

# Random forests
# --------------
# Since caret doesn't allow the number of trees to be tuned,
# we'll set it by using the rule of thumb of 10 × the number of
# predictors (~1500). We'll tune mtry and min.node.size and
# leave splitrule set to "variance." For mtry, the default for
# regression trees is 1/3 × the number of predictors (~47); so,
# we will tune around this value.

# Set tuning parameters
rf.params <- expand.grid(mtry = c(10, 20, 30, 40, 50, 60, 70,
                                  80, 90, 100),
                         splitrule = "variance",
                         min.node.size = c(5, 10))

# Build model
set.seed(723)
rf.model <- train(HeatForm ~ ., data = heat.train,
                  method = "ranger", trControl = cv.params,
                  tuneGrid = rf.params, num.trees = 1500,
                  importance = "impurity", num.threads = 12)

# Analyze results; compute test-set metrics
rf.model$results
write.csv(rf.model$results, file = "rf.csv", row.names = FALSE)

# Best hyperparameter values: mtry = 30, min.node.size = 5

rf.preds <- predict(rf.model$finalModel, data = heat.test)
rf.err <- heat.test$HeatForm - rf.preds$predictions
sqrt(mean(rf.err^2)) # RMSE
mean(abs(rf.err)) # MAE

# Variable importance
rf.vi <- rf.model$finalModel[[6]]
rf.vi <- sort(rf.vi, decreasing = TRUE)
barchart(rf.vi[25:1], xlab = "Variable Importance") # plot top 25

# Boosted decision trees
# ----------------------
# For boosted decision trees, we'll use gbm in caret. We'll tune
# the number of trees, the interaction depth, and the shrinkage and
# leave n.minobsinnode to the default value.

# Set tuning parameters
boost.params <- expand.grid(n.trees = 1500,
                            interaction.depth = c(3, 5, 7),
                            shrinkage = c(0.1, 0.5, 1),
                            n.minobsinnode = 10)
```

```
# Build model
set.seed(723)
boost.model <- train(HeatForm ~ ., data = heat.train,
                     method = "gbm", trControl = cv.params,
                     tuneGrid = boost.params,
                     distribution = "gaussian")

# Analyze results; compute test-set metrics
boost.model$results
write.csv(boost.model$results, file = "boost.csv",
          row.names = FALSE)

# Best hyperparameter values: interaction.depth = 7, shrinkage = 0.1

boost.preds <- predict(boost.model$finalModel, newdata = heat.test,
                       n.trees = 1500)
boost.err <- heat.test$HeatForm - boost.preds
sqrt(mean(boost.err^2)) # RMSE
mean(abs(boost.err)) # MAE

# Variable importance
summary(boost.model$finalModel, plotit = FALSE)

boost.vi <- summary(boost.model$finalModel, plotit = FALSE)
boost.vi.vec <- boost.vi$rel.inf
names(boost.vi.vec) <- boost.vi$var
barchart(boost.vi.vec[25:1], xlab = "Variable Importance")

# Clean up environment
rm(rf.params, boost.params)

# ======================================================================
# Other figures
# ======================================================================
# Histograms of the test-set errors
h1 <- ggplot(NULL, aes(x = fwd.err)) +
      geom_histogram(binwidth = 0.1,
                     fill = "white",
                     color = "black") +
      xlim(-1.5, 1.5) + ylim(0, 1500) +
      xlab("Test-Set Error") + ylab("Count") + theme_bw() +
      annotate("text", x = -1.5, y = 1500,
               label = "(a) Forward",
               size = 4, hjust = 0)
h2 <- ggplot(NULL, aes(x = bwd.err)) +
      geom_histogram(binwidth = 0.1,
                     fill = "white",
                     color = "black") +
      xlim(-1.5, 1.5) + ylim(0, 1500) +
      xlab("Test-Set Error") + ylab("Count") + theme_bw() +
      annotate("text", x = -1.5, y = 1500,
               label = "(b) Backward",
               size = 4, hjust = 0)
h3 <- ggplot(NULL, aes(x = step.err)) +
      geom_histogram(binwidth = 0.1,
                     fill = "white",
                     color = "black") +
```

```
      xlim(-1.5, 1.5) + ylim(0, 1500) +
      xlab("Test-Set Error") + ylab("Count") + theme_bw() +
      annotate("text", x = -1.5, y = 1500,
               label = "(c) Stepwise",
               size = 4, hjust = 0)
h4 <- ggplot(NULL, aes(x = rr.err)) +
      geom_histogram(binwidth = 0.1,
                     fill = "white",
                     color = "black") +
      xlim(-1.5, 1.5) + ylim(0, 1500) +
      xlab("Test-Set Error") + ylab("Count") + theme_bw() +
      annotate("text", x = -1.5, y = 1500,
               label = "(d) Ridge Regression",
               size = 4, hjust = 0)
h5 <- ggplot(NULL, aes(x = lasso.err)) +
      geom_histogram(binwidth = 0.1,
                     fill = "white",
                     color = "black") +
      xlim(-1.5, 1.5) + ylim(0, 1500) +
      xlab("Test-Set Error") + ylab("Count") + theme_bw() +
      annotate("text", x = -1.5, y = 1500,
               label = "(e) Lasso",
               size = 4, hjust = 0)
h6 <- ggplot(NULL, aes(x = tree.err)) +
      geom_histogram(binwidth = 0.1,
                     fill = "white",
                     color = "black") +
      xlim(-1.5, 1.5) + ylim(0, 1500) +
      xlab("Test-Set Error") + ylab("Count") + theme_bw() +
      annotate("text", x = -1.5, y = 1500,
               label = "(f) Decision Tree",
               size = 4, hjust = 0)
h7 <- ggplot(NULL, aes(x = rf.err)) +
      geom_histogram(binwidth = 0.1,
                     fill = "white",
                     color = "black") +
      xlim(-1.5, 1.5) + ylim(0, 1500) +
      xlab("Test-Set Error") + ylab("Count") + theme_bw() +
      annotate("text", x = -1.5, y = 1500,
               label = "(g) Random Forests",
               size = 4, hjust = 0)
h8 <- ggplot(NULL, aes(x = boost.err)) +
      geom_histogram(binwidth = 0.1,
                     fill = "white",
                     color = "black") +
      xlim(-1.5, 1.5) + ylim(0, 1500) +
      xlab("Test-Set Error") + ylab("Count") + theme_bw() +
      annotate("text", x = -1.5, y = 1500,
               label = "(h) Boosting",
               size = 4, hjust = 0)
ggarrange(h1, h2, h3, h4, h5, h6, h7, h8,
          ncol = 3, nrow = 3)
```

C. Training-Set Performance Metrics for Forward Selection and Backward Elimination

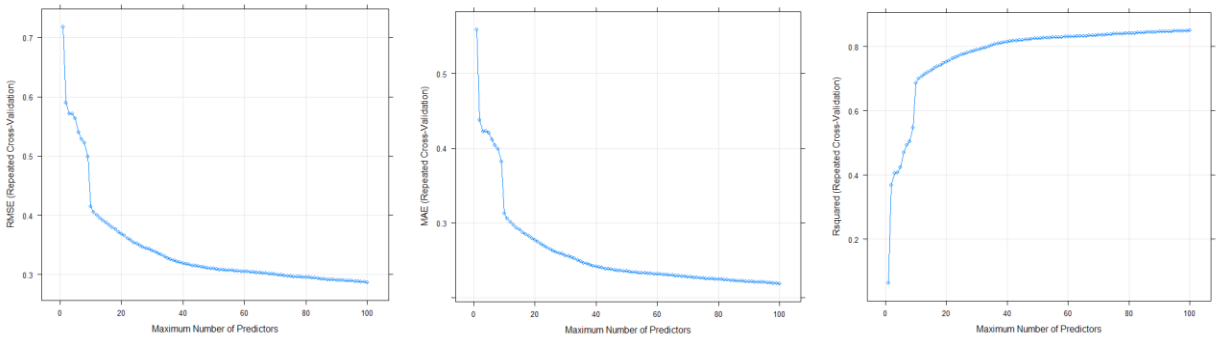

Figure C.1. RMSE, MAE, and R^2 versus the number of predictors for forward selection. There is minimal improvement in performance with more than 50 predictors.

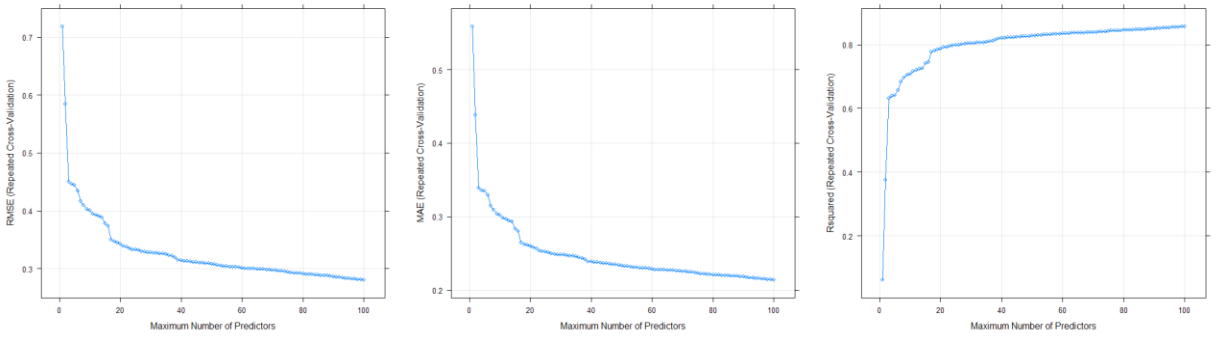

Figure C.2. RMSE, MAE, and R^2 versus the number of predictors for backward elimination. There is minimal improvement in performance with more than 50 predictors.