Supporting Information for

Machine Learning and Statistical Analysis for Materials Science: Stability and Transferability of Fingerprint Descriptors and Chemical Insights

Application of Machine Learning and Statistics to Materials Science

The principal axes of development in this area can be categorized under two sub-categories of the forward problem involving accelerating materials property calculation [3] and the inverse problem of new materials discovery or design. We refer to the problem of accelerating materials property calculation using Machine Learning as the forward problem and the process of materials discovery and materials design from the desired Key-Performance Indicator (KPI) as the inverse problem. The forward problem of accelerating materials property calculation, is important because the model gives valuable knowledge for getting the inverse problem right.

In organic chemistry the approach has a long history generally known as QSAR (Quantitative Structure Activity Relationships) or QSPR (Quantitative Structure Property Relationships), where the P may refer to a large diversity of properties for a large diversity of materials [4, 5].

In a database containing the N set of materials, the set of descriptors $D \in \Re^{N\times P}$ and the KPI $\boldsymbol{y} \in \Re^{N\times 1}$ for each of the materials, what we define by the forward problem is the process of mapping the descriptors to the KPI for accelerating the materials property calculation. In short, the question that we are addressing is:

Can we build a Machine Learning model, $\widehat{\mathcal{M}}$, which is but an estimate of the true model $\mathcal{M}$, such that the KPI property can be directly calculated on-the-fly using only a few descriptors?

As was shown in Figure 1, the true model $\mathcal{M}$, maps the descriptors $D \in \Re^{N\times P}$ for a set of materials (C) in the database to the application specific KPI, and is mathematically defined as $\mathcal{M}: D \mapsto \boldsymbol{y}$. Some of these descriptors and the KPIs are generated by using first principle DFT calculations, for each of the material C in the database. From the dataset it is difficult to determine the true model, $\mathcal{M}$, directly given the limited amount of data and an approximation, $\widehat{\mathcal{M}}$, of it is built using Machine Learning.

The advantage of this is two-fold:

1. For new compounds not already in the database, this reduces the time for calculations, sometimes from hours to second and can be done on the fly.
2. When the estimated model $\widehat{\mathcal{M}}$ is close to the true model $\mathcal{M}$, $\widehat{\mathcal{M}}$ can now be used for the inverse problem of new materials discovery or design.

Bootstrap Projected Gradient Descent (BoPGD) Method:

1. Introduction

In Machine Learning, the Least-Square Linear Regression problem models the "best" linear relationship between the output (also called response) variable and the input variables (also called features/predictors/covariates), given samples containing both. Formally, given a response vector $\boldsymbol{y} \in \Re^{N\times 1}$ a matrix $X \in \Re^{N\times P}$ of predictor variables, a true underlying coefficient vector $\beta \in \mathbb{R}^p$ and an $\mathrm{n} \times 1$ error/noise vector ϵ, the linear model can be expressed as:

$$y = X\beta + \epsilon \tag{1}$$

The objective of Ordinary Least-Square Linear Regression (OLS) is to find out a value of β which minimizes the mean squared error i.e.,

$$\hat{\beta}_{OLS} \in \underset{\beta\in\mathbb{R}^p}{\operatorname{argmin}} \frac{1}{2n} \|y - X\beta\|_2^2 \tag{2}$$

For high-dimensional data, model interpretation becomes a challenging task. That is why Subset Selection (also called Feature Selection) methods are often employed to obtain a sparse yet most significant subset of features that affect the output. In literature, this problem of least square linear regression with feature selection is

popularly known as the Sparse Linear Regression (SPL) problem which alongside minimizing the empirical risk also outputs a sparse coefficient vector β as captured in the following optimization problem:

$$\hat{\beta}_{SPL} \in \underset{\beta \in \mathbb{R}^p : \|\beta\|_0 \leq s}{\operatorname{argmin}} \frac{1}{2n} \|y - X\beta\|_2^2 \quad (3)$$

However, optimal estimation of this problem is infeasible because of non-convexity and NP-hardness. One way to circumvent this is to make the problem convex by relaxing the constraint (since convex problems are easy to solve and they have a unique global minima). A bunch of Iterative Shrinkage Thresholding Algorithms (ISTA) [29] [7] exists which do convex-relaxation of SPL, and possibly the most popular among them is LASSO [7] which does least square regression with regularization/penalization by the ℓ_1 norm. Lasso precisely solves for a β s.t:

$$\hat{\beta}_L \in \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \quad (4)$$

where $\lambda \geq 0$ is a tuning parameter (also called regularization/penalization factor). Depending on the value of λ, the Lasso estimate will have many coefficients set to zero, because of the nature of ℓ_1 penalty. The features corresponding to the non-zero coefficients, also called the support is regarded as the active set of predictors which captures the maximum information in the linear relationship between y and X. Although Lasso is a popular choice when it comes to feature selection due to its model consistency [30, 31, 32] behavior, however, there exists scenarios where strong correlation among the predictors bring instability in Lasso estimate i.e., Lasso tends to select different variables (from the group of correlated or linearly dependent variables) in different independent iterations. A bunch of convex-relaxation based methods (viz. Elastic-net [33], Clustered Lasso [34], CRL [35], BoLasso [36]) have been proposed to solve this problem, however, all of them have slower convergence rate since the optimization problems that they end up solving is non-smooth. The other way of solving this problem as proposed by Jain et. al. [37] is to apply Projected Gradient Descent (PGD) (also called Iterative Hard Thresholding (IHT)) based update rules, wherein at each iteration till convergence, the gradient descent update is projected onto the underlying non-convex feasible set. This is the basis of our algorithm, Bootstrap Projected Gradient Descent (BoPGD) [25] which offers the fastest and consistent solution to the above problem.

2. Bootstrap Projected Gradient Descent (BoPGD)

Given samples containing the input and output variables, the goal of BoPGD [25] is to output a sparse linear model which satisfies the following:

- The model is a "good fit" to the data
- The model can be interpreted easily
- It is consistent when the predictors are highly correlated

Our algorithm scales well under the high dimensional setting too, and has faster convergence rate compared to other existing methods.

There are three key ingredients in BoPGD [25]:

- Clustering
- Projected Gradient Descent
- Bootstrapping

The first step in BoPGD is clustering based on sample correlation among the predictors. This step essentially takes care of the inter-feature correlation which is the root cause of inconsistency in LASSO [7]. The covariates which are strongly correlated are grouped into the same cluster and those which are weakly correlated end up in different clusters. We use bottom-up agglomerative (hierarchical) clustering algorithm to compute the clusters. We then create a representative feature from each cluster as follows:

$$\bar{X}^{(r)} = \frac{1}{|G_r|} \sum_{j \in G_r} X^{(j)} \ , \ r = 1, 2, ..., q \tag{5}$$

where G_r denotes the r^{th} cluster and $\mathrm{X}^{(j)}$ is the j^{th} predictor belonging to G_r. We stack all these cluster representatives column wise and create a Cluster Representative Matrix which forms the input feature matrix for our next step.

The next step involves using a simple Projected Gradient Descent based update rule, as explained in Jain et. al's work [37], on the Cluster Representative Matrix. This step requires the involvement of a hyper-parameter s, the sparsity value which governs the selection of s top most cluster representatives (in terms of their effect on the output variable). We try out a range of values for s and choose the one which corresponds to the minimum average cross-validation error.

The final step is called Bootstrapping. Here we use a voting scheme to eliminate the effect of noise (if any). The idea of how to generate a bootstrap sample is explained in Tibshirani et. al's work [27]. We generate m bootstrapped replicas of the given dataset and run clustering and projected gradient descent on each of these m copies of $\mathrm{X} \in \mathbb{R}^{n \times p}$ with the sparsity parameter s chosen in the second step. We then take the intersection of the supports across these m copies to choose the surviving set of cluster representatives. The variables belonging to those clusters are treated as the most significant variables. Using these cluster representatives we identify the clusters and the corresponding features present in each of them. We consider the union of all these features from the selected clusters as the sparse yet most significant subset of features affecting the output linearly. The pseudo code of BoPGD is presented in Algorithm 1.

Algorithm 1: BoPGD

Input: data $(X, y) \in \mathbb{R}^{n \times (p+1)}$, number of bootstrap replicates m

Steps:

1. Perform hierarchical agglomerative clustering based on correlation among the covariates in X. Let us denote it by cls i.e., $\mathrm{cls(i)}$ refer to the i^{th} cluster.

2. For different values of sparsity s, do:

   2.1. $\theta = \mathbf{PGD}(\mathrm{X}, \mathrm{y}, \mathrm{cls}, \mathrm{s})$

   2.2. $\mathrm{cls}_{sel} = cls(\text{non-zero indices of } \theta)$

   2.3. Selected features, $\mathrm{f}_s = \bigcup_i cls_{sel}(i)$

   2.4. $[\omega, test_{err}] = \mathbf{OLS}(\mathrm{X}(f_s), \mathrm{y})$

3. Let $\mathrm{s}^* = \underset{s}{\operatorname{argmin}} \ test_{err}$

4. For k= 1 to m, do:

   4.1. Generate Bootstrap samples (X^k, y^k)

   4.2. $\theta^k = \mathbf{PGD}(X^k, y^k, cls, s^*)$

4.3. $\mathrm{J}^k = \{\mathrm{j} : \theta_j^k \neq 0\}$ i.e, $\mathrm{supp}(\theta^k)$

5. Compute $cls_{hat} = \bigcap_{k=1}^{m} J^k$

6. Extract features, $f_s = \bigcup_i cls_{hat}(i)$

7. Report $[\omega_{hat},\ test_{err}] = \mathbf{OLS}(\mathrm{X}(f_s),\ \mathrm{y})$

PGD:

Input: X, y, cls, s, Least Square Linear Regression Cost function f

Steps:

1. Compute Cluster Representative Matrix X_{cls}.

2. t = 0.

3. $\theta_t = \mathbf{OLS}(X_{cls},\ \mathrm{y})$

4. while not converged do:

4.1. $\theta_{t+1} = P_s(\theta_t - \alpha\nabla_\theta f(\theta_t))$, where $\mathrm{P}_s(\theta)$ retains only the top s values of θ

4.2. $\mathrm{t} = \mathrm{t} + 1$

5. end while

6. Report θ_t

Descriptor Fingerprinting by BoPGD Algorithm

There are *N* material candidates for the descriptor set $D \in \Re^{N\times P}$ of dimension *P* and for the KPI/property $\boldsymbol{y} \in \Re^{N\times 1}$. As explained in the Introduction section, the set of descriptors *D* given in the database, and their linear and nonlinear combinations produced by engineering them, to give the new extended bag of engineered descriptor $X_E \in \Re^{N\times R}$, $R \gg P$ and $X \subseteq X_E$. As explained in the Introduction Section, the KPI and the engineered descriptors can be related as:

$$M: \boldsymbol{y} = \beta\, f(\Phi(D)) + \boldsymbol{\epsilon}$$

Where the engineered descriptor is obtained from the original descriptor space $X_E := \Phi(D)$. However, there occur only a smaller number of descriptors, say *K*, which are useful for predicting the KPI (Binding Energy) s. t. $K < P$ and $P \ll R$. We introduce the Bootstrapped Projected Gradient Descent (BoPGD) algorithm, which is represented by the functional form *f*. It is used here to choose the sparsest representation of the descriptors and its derivatives from the original or the engineered descriptor space. To summarize the algorithm,

Step 1: BoPGD takes up the descriptor engineered dataset $(X_E, \boldsymbol{y})$ and clusters the descriptors together depending on their cross-correlation strength. The resulting clustered dataset is represented as $(X_C, \boldsymbol{y})$ s.t $X_c \in \Re^{N\times M}$ and $M \ll P$.

Step 2: Predict the minimum sparsity or support of the clustered descriptor space (i.e. set of dominant clusters that affect the KPI) $K = \text{supp}(\beta), s.t. K \in [1, 2, \dots, M],\ K < P \ll M < R$ using a simple projected gradient descent-based update.

Step 3: For every bootstrapping trial, T, iteratively, fit the reduced descriptor space X_c to the KPI y so as to estimate the set of coefficients $\beta \in \Re^M$ that best maps X_c to $\boldsymbol{y}$ such that $y \approx \hat{\beta} X_c$. At every iteration, limit the number of $\hat{\beta}$ to the non-zero values or set $\|\hat{\beta}\|_0$=K.

Step 4: Group the T-bootstrapped coefficients to a matrix $B \in \Re^{T \times K}$ that consists of the selected coefficients from each bootstrap iterations.

Step 5: Calculate the final ranked descriptor cluster $\hat{\beta}$ from the coefficient matrix B by taking the most recurring descriptor cluster, or the soft intersections, from the T trials. Usually, the estimated coefficient has a cardinality $\|\hat{\beta}\| \leq K$.

Step 6: We choose one representative descriptor from the down selected descriptor cluster that is closest to the centroid.

This representative descriptor becomes the unique fingerprint for predicting the target property or KPI using a linear or nonlinear function to our dataset. The clustering of the descriptors together ensures that the descriptors that are collinear are cumulated together into a single representative descriptor. In order to obtain the most stable fingerprint descriptors, we take several bootstrap iterations of the description selection process results and select the ones that most "frequently" appear. This ensures that any uncertainty in the selection process due to noise in the dataset or due to the limited size of the dataset is smoothened out.

For the BoPGD, the increase in the number of features beyond K did not decrease the error rates in the linear or the nonlinear fit. The maximum error is 0.3eV between the predicted and the calculated BE and minimum error is 0eV.

The computational time complexity for the LASSO descriptor selection algorithm, measures as $\mathcal{O}(\mathrm{M}^2\mathrm{N} + \mathrm{M}^3)$ when implemented using the LARS algorithm [7], where the number of columns is M and the numbers of samples is N. For the BoLASSO algorithm [36], involving bootstrapped trials T of the LASSO algorithm, the complexity is $\mathcal{O}(\mathrm{TM}^2\mathrm{N} + \mathrm{TM}^3)$. For the BoPGD algorithm, the hierarchical clustering algorithm has an independent order of $\mathcal{O}\big(\mathrm{M}^2\log(\mathrm{M})\big)$ and for T bootstraps, the complexity is $\mathcal{O}(\mathrm{TK}^2\mathrm{N} + \mathrm{TK}\log(\mathrm{K}))$ with $\mathrm{K} \ll \mathrm{M}$.

Principal Component Analysis:

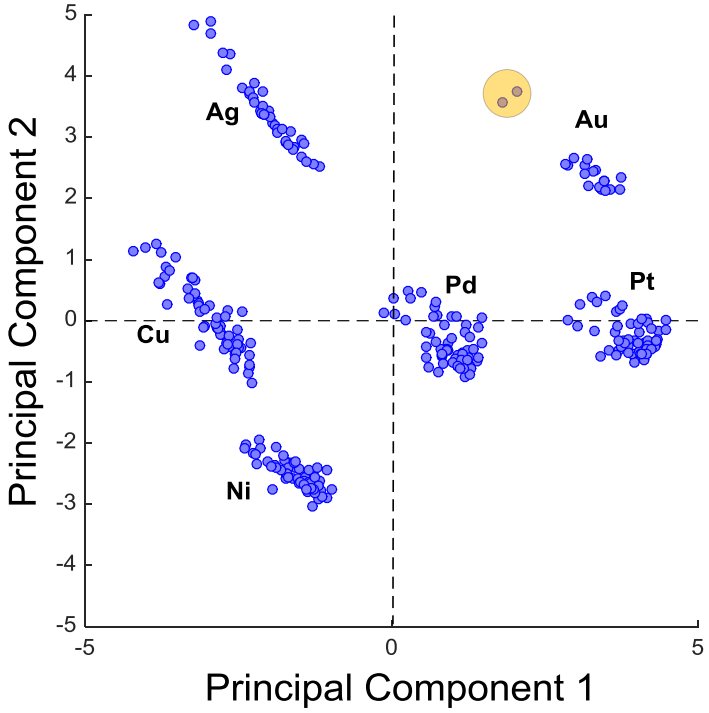

Figure S1: Score plot of the first two principal components. There is a clear clustering among the different data points indicative of the substrate with positive first principal component scores for Pd, Pt and Au, and negative first principal component scores for Ni, Cu and Ag. The two outliers of the substrate Au namely "Au-La@Au" and "Au-Y@Au" are shown highlighted by the yellow circle.

Table of Contents Figure:

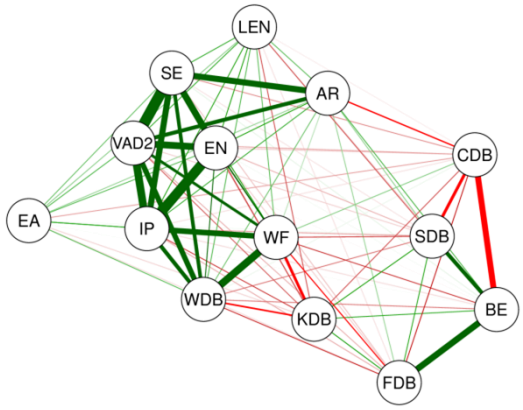

Table of content figure