Predicting Transparent Conductors

Adebisi Afolalu, 120693551
Batuhan Yildirim, 170749381
Tanmaiyii Rao, 140361229
Blanca Herrero Garcia, 170604806

April 2, 2018

Contents

1 Introduction

1.1 Problem statement and hypothesis.

Improving the properties of materials that are intrinsically connected to the generation and utilization of energy is crucial if we are to mitigate environmental damage due to a growing global demand. Transparent conductors are an important class of compounds that are both electrically conductive and have a low absorption in the visible range, which are typically competing properties. A combination of both of these characteristics is key for the operation of a variety of technological devices such as photovoltaic cells, light-emitting diodes for flat-panel displays, transistors, sensors, touch screens, and lasers. However, only a small number of compounds are currently known to display both transparency and conductivity suitable enough to be used as transparent conducting materials.

Aluminum (Al), gallium (Ga), indium (In) sesquioxides are some of the most promising transparent conductors because of a combination of both large bandgap energies, which leads to optical transparency over the visible range, and high conductivities. These materials are also chemically stable and relatively inexpensive to produce. However, the main limitation in the design of compounds is that identification and discovery of novel materials for targeted applications requires an examination of enormous compositional and configurational degrees of freedom (i.e., many combinations of x, y, and z).

To avoid costly and inefficient trial-and-error of synthetic routes, computational data-driven methods can be used to guide the discovery of potentially more efficient materials to aid in the development of advanced (or totally new) technologies. In computational material science, the standard tool for computing these properties is the quantum-mechanical method known as density-functional theory (DFT). However, DFT calculations are expensive, requiring hundreds or thousands of CPU hours on supercomputers for large systems, which prohibits the modeling of a sizable number of possible compositions and configurations. Data-driven models offer an alternative approach to efficiently search for new possible compounds in targeted applications but at a significantly reduced computational cost. [2]

This project aims to accomplish this goal by applying data analytics models (supervised regression models) for the prediction of two target properties: the formation energy, -which is an indication of the stability of a new material- and the bandgap energy – which is an indication of the potential for transparency over the visible range – to facilitate the discovery of new transparent conductors and allow for advancements in the mentioned technologies.

2 Python Libraries

2.1 Pandas

Pandas was utilised in this project to enable analysis and manipulation of the dataset. Pandas is an open-source Python library built on top of the NumPy library. It provides easy–to–use high-performance data structures as well as ability to store columns of different data types. Data can be stored as a "Series" or a "data frames" data structure.

Pandas contains functionality for importing and exporting data between in-memory data structures and different formats (e.g. CSV, Excel and SQL databases). In this project, the dataset was organised in a table in an external CSV file. The CSV was loaded into a data frame by using Pandas functions. [8]

Once the data was loaded in the Python environment, the data could be easily prepared, cleaned, manipulated and sliced as required. As a result, it was easier to run the machine learning algorithms and carry out the analysis on the data.

2.2 Matplotlib

Matplotlib was used for all the data visualisation in the project. It is a Python 2D plotting library providing functionality to generate high quality figures (such as plots, bar charts and histograms). The figures outputted only required a few lines of Python code and have been included throughout this report. [7]

2.3 Scikit learn

Scikit learn was vital for carrying out a majority of the data mining tasks. This open-source Python library (built on NumPy, SciPy and Matplotlib) contains machine learning algorithms. All the machine learning algorithms required was invoked with a few lines of code, without the need to implement the algorithms from scratch.[12]

The library includes supervised learning models for classification and regression. The regression models SVR, Ridge Regression and XGBoost was utilised to output real numbers for the predictions. It also includes unsupervised learning models for clustering and dimensionality reduction. The dimensionality reduction algorithms PCA and feature selection was used to improve the performance of the regression models.

In addition, Scikit learn possesses functionality for model selection and preprocessing. Cross validation was implemented from model selection sub-package for finding the optimum parameters of the models. Normalisation of all the features was achieved with the data scaling functions. This was available from the preprocessing sub-package.

3 Background Information

3.1 RMSLE (Root Mean Squared Logarithmic Error)

Root Mean Squared Logarithmic Error (RMSLE) is a technique to compute the difference between the values predicted by the machine learning model and the actual values. RSMLE is an application of MSE (Mean Squared Error). MSE incorporates both the variance and the bias of the predictor. RMSE (Root Mean Squared Error) is the square root of MSE. In the case of RMSLE , the log of the predictions and the actual values is taken. So basically what changes is the variance that you are measuring. RMSLE is generally used when you do not want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers. [11]

The RMSLE for a single column calculated as

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

where:
n is the total number of observations
p_i is your prediction
a_i is the actual value
$\log(x)$ is the natural logarithm of x

The final score is the mean of the RMSLE over all columns (in this case, 2).

3.2 SVM (Regression)

SVM classifiers are binary or discriminating models, working on two classes of differentiation. Support Vector Machine can be applied not only to classification problems but also to the case of regression. Their main task is basically to discriminate against new observations between two classes.

During the learning phase, these classifiers project the observations in a multidimensional space called decisional/feature space and build a separation surface called decision boundary that divides this space into two areas of belonging. The linear case (simplest case), the decision boundary will be represented by a plane (in 3D) or by a straight line (in 2D). In more complex cases the separation surfaces are curved shapes with increasingly articulated shapes. SVMs can be used both in regression with SVR (Support Vector Regression) and in classification with SVC (Support Vector Classification).

In the case of Support Vector Regression (SVR) a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea remains the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated. [17]

In order to implement SVR in Python, sklearn.svm.SVR has been used.

```
class sklearn.svm. SVR (kernel='rbf', degree=3, gamma='auto', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1,
shrinking=True, cache_size=200, verbose=False, max_iter=-1) ¶
```

Kernel specifies the kernel type to be used in the algorithm. It could be one of "linear", "poly", "rbf", "sigmoid", "precomputed" or a callable. On this project, the kernel is "poly". [16]

3.3 Linear Regression

Linear regression is a simple yet very powerful machine learning algorithm. Linear regression is widely used in different supervised machine learning problems. In regression analysis, a number of predictor variables and a continuous response variable are given, and we try to establish a relationship between those variables that allows us to predict an outcome. (Ref: Lecture slide, skLearn,) A linear regression line has an equation of the form

$$Y = a + bX$$

where X is the explanatory variable and Y is the dependent variable and b is the slope.

The most common method for fitting a regression line is the method of least-squares. This method calculates the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line (if a point lies on the fitted line exactly, then its vertical deviation is 0). [6]

3.3.1 Ridge Regression

Ridge Regression is a type of linear regression. It is used for the analysis of multiple regression data that suffer from multicollinearity. Least squares estimates are unbiased, but their variances are large so they may be far from the true value, when multicollinearity occurs. Ridge regression reduces the standard errors, by adding a degree of bias to the regression estimates. It is hoped that the net effect will be to give estimates that are more reliable. [10]

It is similar to least squares but shrinks the estimated coefficients towards zero. Given a response vector $y \in R_n$ and a predictor matrix $X \in R_{n \times p}$, the ridge regression coefficients [1] are defined as

$$\begin{aligned}\hat{\beta}^{\text{ridge}} &= \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \\ &= \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_2^2}_{\text{Penalty}}\end{aligned}$$

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2–norm. In sci-kit learn, the function is given as the following :

```
class sklearn.linear_model.Ridge(alpha=1.0, fit_intercept=True, normalize=F
alse, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
```

where alpha is a parameter for regularization. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. [18]

Here $\lambda \geq 0$ is a a which controls the strength of the penalty term. [1] Note that:

When $\lambda = 0$, we get the linear regression estimate.

When $\lambda = \infty$, we get $\widehat{\beta}^{ridge} = 0$.

For λ in between, we are balancing two ideas: fitting a linear model of y on X, and shrinking the coefficients.

Ridge Regression Model

Following the usual notation, as stated by https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf suppose the regression equation is written in matrix form as $Y = XB + e$ where Y is the dependent variable, X represents the independent variables, B is the regression coefficients to be estimated, and e represents the errors are residuals. In ordinary least squares, the regression coefficients are estimated using the formula

$$\underline{\widehat{B}} = (X'X)^{-1}X'\underline{Y}$$

Note that since the variables are standardized, $X'X = R$, where R is the correlation matrix of independent variables. These estimates are unbiased so that the expected value of the estimates are the population values. That is,

$$E(\underline{\widehat{B}}) = \underline{B}$$

The variance-covariance matrix of the estimates is

$$V(\underline{\widehat{B}}) = \sigma^2 R^{-1}$$

And since we are assuming that the y are standardized, $\sigma^2 = 1$. From the above, we find that

$$V(\widehat{b}_j) = R^{jj} = \frac{1}{1 - R_j^2}$$

where R_j^2 is the R–squared value obtained from regression X_j on the other independent variables. We see that as the R–squared in the denominator gets closer and closer to one. Now, ridge regression proceeds by adding a small value, k, to the diagonal elements of the correlation matrix. (Thus the name ridge regression since the diagonal of ones in the correlation matrix can be assumed to be a ridge.) That is,

$$\underline{\widetilde{B}} = (R + kI)^{-1}X'\underline{Y}$$

k is a positive quantity less than one (usually less than 0.3).

The amount of bias in this estimator is given by

$$E(\underline{\widetilde{B}} - \underline{B}) = \left[(X'X + kI)^{-1}X'X - I\right]\underline{B}$$

and the covariance matrix is given by

$$V(\underline{\widetilde{B}}) = (X'X + kI)^{-1}X'X(X'X + kI)^{-1}$$

It can be shown that there exists a value of k for which the mean squared error (the variance plus the bias squared) of the ridge estimator is less than that of the least squares estimator. Unfortunately, the appropriate value of k depends on knowing the true regression coefficients (which are being estimated) and an analytic solution has not been found that guarantees the optimality of the ridge solution. [10]

Model Application

In this case, we use RSMLE which has been defined before as the error function that we need to calculate instead of MSE.

3.4 GridSearch

Grid search is the process applied when you have a set of models (which differ from each other in their parameter values, which lie on a grid). Each model is trained and evaluated using cross-validation. The best performing model is then selected amongst those. [5]

The grid search in sklearn is **GridSearchCV** which exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter. For instance, the following param_grid:

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
```

specifies that two grids should be explored: one with a linear kernel and C values in $[1, 10, 100, 1000]$, and the second one with an RBF kernel, and the cross-product of C values ranging in $[1, 10, 100, 1000]$ and gamma values in $[0.001, 0.0001]$. [4]

3.5 XGBoost

XGBoost stands for eXtreme Gradient Boosting. It is a library for optimising boosted tree algorithms. The primary goal is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate for large scale tree boosting.

XGBoost implements the gradient boosting decision tree algorithm. Boosting is an ensemble technique where existing models are corrected by the addition of new models. Models are added in sequence until no more modifications can be made. Gradient boosting is a technique where new models are created that predict the errors of prior models and then combined together to determine the final prediction. Since it uses gradient descent algorithm to minimize the loss when adding new models, it is called gradient boosting. This approach supports both regression and classification predictive modelling problems. [19]

3.5.1 Tree Boosting in brief

For a given data set with n examples and m features $D = \{(x_i, y_i)\}(\mid D \mid = n, x_i \in R^m)$), a tree ensemble model (shown in Fig. 1 (left) uses K additive functions to predict the output.

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F},$$

where $F = \{f(x))w_{q(x)}\}(q : R^m \longrightarrow T, w \in R^m)$) is the space of regression trees.

Here q represents the structure of each tree that maps an example to the leaf index and w is the weight vector of each leaf.

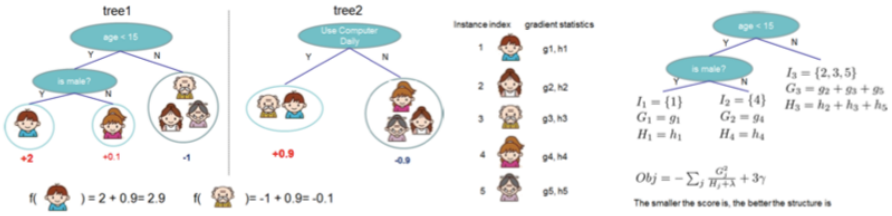


Figure 1: *Left:* The tree ensemble model. *Right:* The structure score calculation of a single tree.

For a given example, the decision rules in the trees (given by q) are used to classify them into the leaves and calculate the final prediction by summing up the scores in the corresponding leaves (given by w). To learn a tree ensemble, the following regularized objective has been optimised:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \;\; \Omega(f) = \gamma T + \lambda \|w\|^2.$$

Here l is a differentiable convex loss function that measures the quality of prediction $y\hat{}i$ on training data, the second term ω measures the complexity of the model to avoid over–fitting.

The model is trained in an additive matter. At each iteration t, we add a new tree to optimize the objective. We can derive a score to measure the quality of a given tree structure q (the derivation is omitted due to space constraints).

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$$

where g_i and h_i are the gradient statistics. This score is like the impurity score for assessing decision trees, except that it is derived for wider range of objective functions. Fig. 1 (right) shows how the score can be calculated: only need to calculate the sum statistics of examples in each leaf node, then apply Eq (1) to get the quality of the tree. A greedy algorithm will iteratively search over the possible split candidates and find the best split to add to the tree until a maximum depth is reached. [20]

4 Description of data

The data was obtained from Kaggle and contains 3000 materials: 2400 for training, and 600 for testing. The features of the data contain properties of the materials that are promising for the development of transparent conductors. The following features are included:

1. Spacegroup – the symmetry group of the configuration of atoms inside the material.
2. Total No. of Atoms - the number of Al, Ga, In, and O atoms inside a unit cell, which is a repeated group of atoms in three dimensions which make up the entire structure of a crystal.
3. Relative percentages of Al, Ga and In atoms inside the unit cell.
4. Lattice Vectors - lv1, lv2 and lv3. Vectors which connect individual atoms to other atoms within the crystal lattice (given in units of *Angstroms* ($10^{10}m$)
5. Lattice Angles - α, β and γ represent the relative angles between atoms in the x, y and z directions respectively.
6. Formation Energy – the energy required to produce a defect in a perfect crystal structure. Hence, formation energy is an important indicator of the stability of a material.
7. Bandgap Energy – the difference in energy between the conduction and valence bands within the material. The bandgap energy determines whether a material is a conductor, semiconductor or an insulator. It is an important property for the optoelectronic properties of a material.

5 Data Preprocessing

5.1 Scaling

The features in the dataset all contain values which lie in different ranges. For example, atom percentages range from $0-1$, while lattice angles range from $0-360$. This is a problem for the algorithms which use gradient descent to learn. If the features all have different ranges, gradient descent may not converge as it will descend quickly for features with small ranges, and slowly for features with large ranges. It will therefore oscillate inefficiently down the optimum when the variables are uneven.

Min-max scaling was used, which scales the values between 0 and 1 based on the min and max of the feature.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

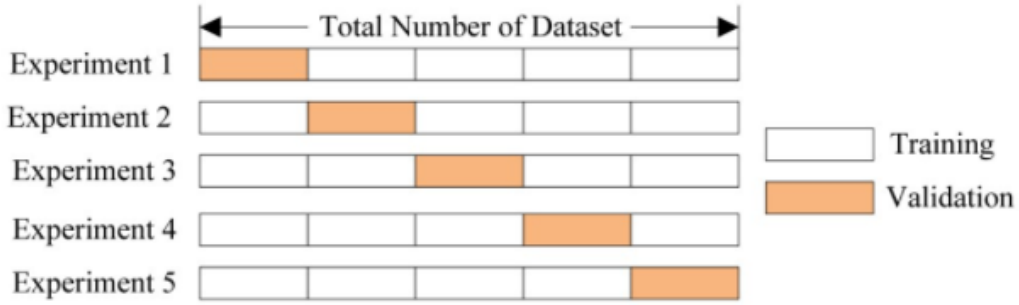
where x is the original value and x' is the normalised value.

6 Implementation

6.1 Cross–Validation

All models were evaluated during training using 10–fold cross–validation to check if the models were overfitting and to evaluate each models performance on the training data. Evaluating the model on the training data using cross-validation also allows hyperparameter tuning.

Cross-validation is performed by partitioning the training data into training and validations sets. The model is trained on the training data, and evaluated on the validation set. In k–fold cross–validation, a model is trained on $k - 1$ of the folds the training data and the resulting model is validated on the remaining part. This process is repeated k times, in a manner which means the model is trained and evaluated on all possible k combinations of training and validation data. For example, in 5–fold cross validation, 5 models will be trained evaluated using 5 different splits of train and validation data, as shown in figure (6.1). This ensures that all of the data is used for training, and all used for evaluation.



The performance of the model is then evaluated by taking the mean of the cross-validations scores. In cases where training data is large, this process is computationally expensive and can take a lot of time to train. In the case of this project where the data was relatively small, this was not a problem. 10-fold cross–validation was used in all models.

6.2 PCA

A function was created to perform PCA on each model and test the predictive accuracy of the models against all possible number of principal components used. The function takes as input the training data, the model and k-value for cross-validation. PCA is performed in a loop with n iterations, where n is the number of features in the dataset. With each iteration PCA is fitted to, and transforms the training data. The resulting PCA data is converted into a pandas dataframe, which is then used to train and evaluate the model using cross-validation.

The function returns two lists which contain predictive accuracies for binding energy and formation energy.

```
# Function for PCA

from sklearn.decomposition import PCA

def PCA_scores(train, model, cv):
    be_score = []
    fe_score = []

    for i in np.arange(1,len(columns)):
        pca = PCA(n_components=i, random_state=9)
        pca_train = pca.fit_transform(train)
        pca_train = pd.DataFrame(pca_train)

        be_score.append(np.mean(cross_val_score(model, pca_train, be, cv=cv, scoring=rmsle_score)))
        fe_score.append(np.mean(cross_val_score(model, pca_train, fe, cv=cv, scoring=rmsle_score)))

    return be_score, fe_score
```

6.3 SVM

The SVM was set up with a polynomial kernel, and gamma and C values of 1. Cross–validation was performed using scikit-learn's *cross_val_score()* function.

```
from sklearn.svm import SVR

svr = SVR(kernel='poly',
          C=1,
          gamma=1)
```

The PCA_scores function is used on the SVR estimator and the PCA scores for binding energy and formation energy are plotted against the number of principal components.

```
# SVM with PCA

svm_be_score, svm_fe_score = PCA_scores(train, svr, 3)
```

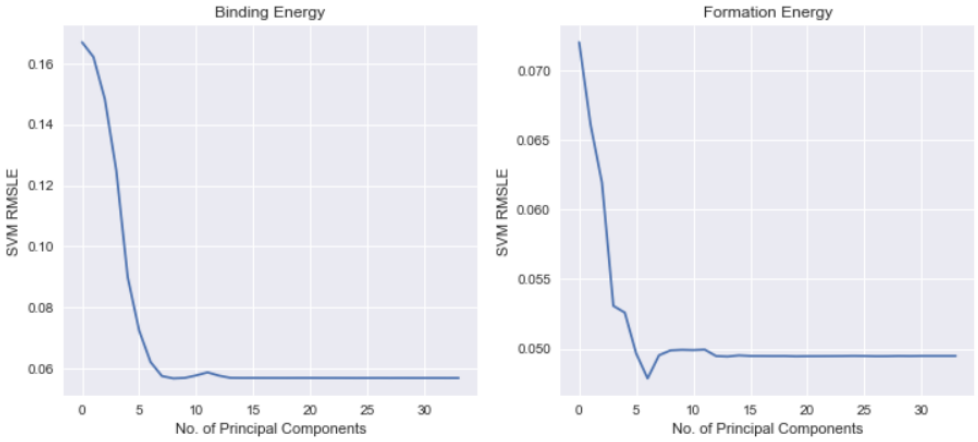


Figure (6.3) shows that the lowest error for both binding and formation energy predictions was achieved using 9 principal components. Thus, the model with 9 principal components was used for the final predictions. The cross validation RMSLE scores for binding energy and formation energy were 0.0541 and 0.0479.

6.4 Ridge Regression

The ridge regression estimator was set with a regularisation alpha of 0.00001, as this achieved the best accuracy.

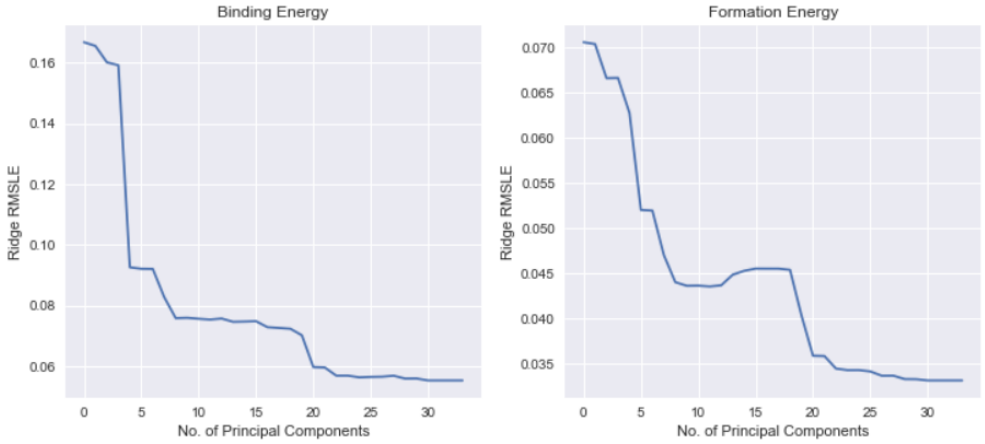
```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=0.00001,
              random_state=9)
```

Once again, PCA was performed iteratively with all possible number of principal components and the results were plotted.

```
# Ridge with PCA

ridge_be_score, ridge_fe_score = PCA_scores(train, ridge, 3)
```



It was found that using 20 principal components achieved the lowest error, so this model was fitted and evaluated with cross-validation. The results were cross-validation scores of 0.0680 and 0.0389 respectively for binding energy and formation energy.

6.5 XGBoost and GridSearch

First, the initial XGB model is set up for GridSearch with the parameters initialised arbitrarily. The hyperparameter grid that GridSearch will iterate through was then set up.

```
import xgboost as xgb

xgb_ = xgb.XGBRegressor(n_jobs=10,
             objective='reg:linear',
             eta=0.1,
             eval_metric='rmse',
             n_estimators=500,
             silent=1,
             seed=90,
             n_stopping_rounds=10)
```

```
from sklearn.grid_search import GridSearchCV

learning_rate_range = [0.01, 0.05, 0.1]
max_depth_range = [3, 4, 5, 6, 7]
min_child_weight_range = [6, 7, 8]
subsample_range = [0.6, 0.7, 0.8, 0.9]
colsample_range = [0.7, 0.8, 0.9]

xgb_param_grid = dict(learning_rate=learning_rate_range,
                      max_depth=max_depth_range,
                      min_child_weight=min_child_weight_range,
                      subsample=subsample_range,
                      colsample_bytree=colsample_range)
```

The parameters found by Gridsearch for predicting binding energy and formation energy are shown in figure (6.5).

```
Binding Energy Parameters:
 {'colsample_bytree': 0.7, 'eta': 0.01, 'learning_rate': 0.05, 'max_dept
h': 3, 'min_child_weight': 8, 'subsample': 0.6}


{'colsample_bytree': 0.9, 'eta': 0.01, 'learning_rate': 0.01, 'max_dept
h': 6, 'min_child_weight': 8, 'subsample': 0.8}
```

Following Gridsearch, as the hyperparameters for predicting the two target variables differed, two XGB models were created and cross-validated.

```
#Binding Energy

xgb_be = xgb.XGBRegressor(n_jobs=10,
              objective='reg:linear',
                          colsample_bytree=0.7,
                          learning_rate=0.05,
                          max_depth=3,
                          min_child_weight=8,
                          subsample=0.8,
                          eta=0.01,
                          eval_metric='rmse',
                          n_estimators=1000,
                          silent=1,
                          seed=90,
                          n_stopping_rounds=10)

#Formation Energy

xgb_fe = xgb.XGBRegressor(n_jobs=10,
              objective='reg:linear',
                          colsample_bytree=0.9,
                          learning_rate=0.01,
                          max_depth=6,
                          min_child_weight=8,
                          subsample=0.8,
                          eta=0.01,
                          eval_metric='rmse',
                          n_estimators=1000,
                          silent=1,
                          seed=90,
                          n_stopping_rounds=10)
```

The cross-validation scores were 0.0511 and 0.0262 respectively.

Table 1: Table with CV Scores for BE and FE for each model

	SVM	Ridge Regression	XGboost
binding energy	0.0541	0.0680	0.0511
formation energy	0.0479	0.0389	0.0262

From the results in Table (1), it can be seen that XGBoost performed the best and achieved the lowest error for predicting both binding energy and formation energy. This is likely due to the use of Gridsearch to exhaustively search for the best hyperparameters, as well as the fact that it is a boosting algorithm.

For this reason, two XGB estimators were created to be trained on all of the training data, and tested using the test data. These estimators were set with the hyperparameters learned by Gridsearch.

A Kaggle RMSLE score of 0.0607 was achieved on the test data by the XGB model. It is unclear if this score is an average of both the binding energy and formation energy scores.

7 Feature Engineering and Dimensionality Reduction

7.1 Feature Extraction

Feature extraction was carried out in this project for dimensionality reduction. It is a feature engineering process, where domain knowledge of the data was used to derive new features. The initial dataset has 11 features, which was used to create 24 new features. This resulted in a new dataset with 35 features.

The new dataset with 35 features requires more memory and processing time, while also causing the regression models used to overfit the training samples. Feature extraction helped alleviate the "curse of dimensionality" problem by combining features. This reduced the number of features required to generalise the data.

Further dimensionality reduction techniques described in the Feature Selection and Principal Component Analysis (PCA) sections was employed. The 35 features were reduced to an optimum subset of 9 features, which included the "lattice_angle_beta_gamma_degree ", "lattice_angle_gamma_alpha_degree " and "lattice_vector_2_ang_/_3 "columns created by feature extraction. [3] The 14 new features were manually extracted with the following code:

```
def new_features(df):
    df['percent_atom_al_ga'] = df['percent_atom_al'] * df['percent_atom_ga']
    df['percent_atom_al_in'] = df['percent_atom_al'] * df['percent_atom_in']
    df['percent_atom_ga_in'] = df['percent_atom_ga'] * df['percent_atom_in']

    df['lattice_vector_1_ang_/_2'] = df['lattice_vector_1_ang'] / df['lattice_vector_2_ang']
    df['lattice_vector_2_ang_/_3'] = df['lattice_vector_2_ang'] / df['lattice_vector_3_ang']
    df['lattice_vector_3_ang_/_1'] = df['lattice_vector_3_ang'] / df['lattice_vector_1_ang']

    df['lattice_angle_alpha_beta_degree'] = df['lattice_angle_alpha_degree'] *
                                            df['lattice_angle_beta_degree']
    df['lattice_angle_beta_gamma_degree'] =        df['lattice_angle_beta_degree'] *
                                            df['lattice_angle_gamma_degree']
    df['lattice_angle_gamma_alpha_degree'] = df['lattice_angle_gamma_degree'] *
                                            df['lattice_angle_alpha_degree']

    df['percent_atom_al_ga_in/lv1'] = (df['percent_atom_al'] + df['percent_atom_ga'] +
                                       df['percent_atom_in']) / df['lattice_vector_1_ang']
    df['percent_atom_al_ga_in/lv2'] = (df['percent_atom_al'] + df['percent_atom_ga'] +
                                       df['percent_atom_in']) / df['lattice_vector_2_ang']
    df['percent_atom_al_ga_in/lv3'] = (df['percent_atom_al'] + df['percent_atom_ga'] +
                                       df['percent_atom_in']) / df['lattice_vector_3_ang']

    df['percent_atom_al_ga_in*alpha'] = (df['percent_atom_al'] + df['percent_atom_ga'] +
                             df['percent_atom_in']) * df['lattice_angle_alpha_degree']
    df['percent_atom_al_ga_in*beta'] = (df['percent_atom_al'] + df['percent_atom_ga'] +
                            df['percent_atom_in']) * df['lattice_angle_beta_degree']
    df['percent_atom_al_ga_in*gamma'] = (df['percent_atom_al'] + df['percent_atom_ga'] +
                            df['percent_atom_in']) * df['lattice_angle_gamma_degree']

    df['lattice_vector_A_B_G_1'] = np.sqrt(df['lattice_angle_alpha_degree']
    * df['lattice_angle_beta_degree'] *df['lattice_angle_gamma_degree']) / df['lattice_vector_1_ang']
    df['lattice_vector_A_B_G_2'] = np.sqrt(df['lattice_angle_alpha_degree']
    * df['lattice_angle_beta_degree'] * df['lattice_angle_gamma_degree']) / df['lattice_vector_2_ang']
    df['lattice_vector_A_B_G_3'] = np.sqrt(df['lattice_angle_alpha_degree']
     * df['lattice_angle_beta_degree'] * df['lattice_angle_gamma_degree']) / df['lattice_vector_3_ang']

    df['lattice_123_A_B'] = (df['lattice_vector_1_ang'] + df['lattice_vector_2_ang'] +
     df['lattice_vector_3_ang']) / (df['lattice_angle_alpha_degree'] * df['lattice_angle_beta_degree'])
    df['lattice_123_B_G'] = (df['lattice_vector_1_ang'] + df['lattice_vector_2_ang'] +
    df['lattice_vector_3_ang']) / (df['lattice_angle_beta_degree'] * df['lattice_angle_gamma_degree'])
    df['lattice_123_G_A'] = (df['lattice_vector_1_ang'] + df['lattice_vector_2_ang'] +
     df['lattice_vector_3_ang']) / (df['lattice_angle_gamma_degree'] * df['lattice_angle_alpha_degree'])

    df['NTA_al_ga_in'] = df['number_of_total_atoms'] * (df['percent_atom_al'] +
                         df['percent_atom_ga'] + df['percent_atom_in'])
    df['NTA_1_2_3'] = df['number_of_total_atoms'] * (df['lattice_vector_1_ang'] +
                         df['lattice_vector_2_ang'] + df['lattice_vector_3_ang'])
    df['NTA_A_B_G'] = df['number_of_total_atoms'] * (df['lattice_angle_alpha_degree'] +
     df['lattice_angle_beta_degree'] + df['lattice_angle_gamma_degree'])

    return(df)
```

7.2 Principal Component Analysis (PCA)

The unsupervised learning algorithm Principal Component Analysis (PCA) is used in this project to explore the amount of variance explained by each column. It uncovers the underlying structure of the data. The columns with the highest variance is the most informative and will improve the predictive accuracy of the regression models implemented. This was vital for dimensionality reduction, as columns with very low variance were disregarded.

PCA works by orthogonally transforming the columns in the dataset. This converts the columns (which are possibly correlated to the classification labels) into a set of columns that are linearly uncorrelated variables called principal components. [9]

The first principal components (denoted as 1) accounts for the largest possible variance and each component that follows has the next highest variability in ranked order.

7.2.1 PCA Implementation

1. Firstly, all the columns is normalised with the following line of code, using Sklearn library:

```
# data scaling
X_std = sklearn.preprocessing.StandardScaler().fit_transform(X)
```

2. Then, the eigendecomposition is computed on the covariance matrix of the features. This gives the columns as eigenvectors with its corresponding eigenvalues.

```
# eigendecomposition on covariance matrix
cov_mat = np.cov(X_std.T)

eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
fit_transform(X)
```

3. The eigenvectors and the coresponding eigenvalues was sorted by the eigenvalues in deceding order.

```
# make a list of (eigenvalue, eigenvector) tuples
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in range(len(eigen_vals))]
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in range(len(eigen_vals))]

# sort the (eigenvalue, eigenvector) tuples in descending order
eigen_pairs.sort()
eigen_pairs.reverse()
```

4. The eigenvalues corresponds to the explained variance, which was converted to a percentage.

```
#calculates the explained variance as a percentage
total = sum(eigen_vals)
exp_var = [(i / total)*100 for i in sorted(eigen_vals, reverse=True)]
```

5. The eigenvectors (principal component) was plotted against the corresponding explained variance percentage in a barchart. The first principal component was labelled 1, the second prinicpal component was labelled 2 and so on.

```
#generates a barchart of explained variance against PC
plt.bar(d.keys(), exp_var)
plt.xlabel('Principal Components (PC)')
plt.ylabel('Explained Variance %')
plt.title('Principal Component Analysis (PCA)')
plt.show()
```

7.2.2 PCA Results

The Principal Component Analysis represented in the bar chart below, shows that 9 out of 35 features has 96.7% of the explained variance. This suggests that the remaining features will have a negligible amount of information. As a result, we only trained the regression models with the first 9 principal components. This improved the efficiency and accuracy of the models.

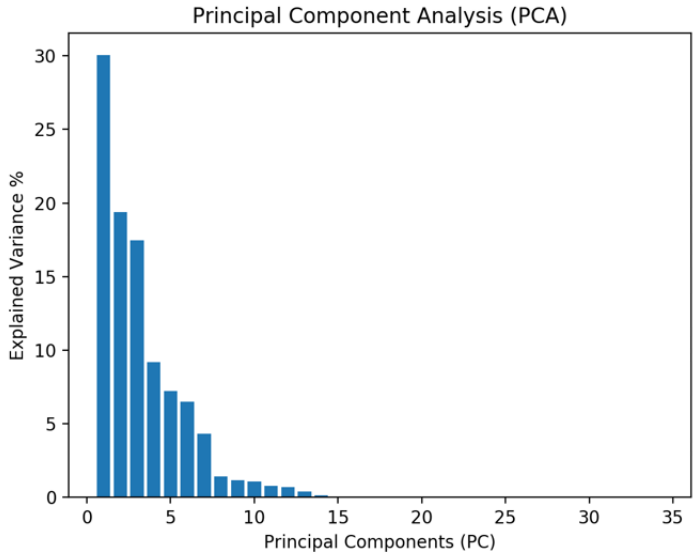


Figure 1: PCA Bar chart

7.3 Feature Selection

Feature selection was used in this project to select the optimum subset of features for training the SVR, ridge regression and XGBoost models. The optimum subset was found by removing the redundant or irrelevant columns from the data, leaving the most informative predictors. The feature selection process made the models simpler to understand, shortened the training times, reduced overfitting and avoided the "curse of dimensionality".

Backwards selection was favoured over forward selection because it takes into account the combination of the column. Backwards selection was implemented with the Recursive Feature Elimination (RFE). SVR estimator was trained on the set of features and the importance was measured with the coef attribute. The least important feature with a coef closest to zero was pruned from the set of features. This process was recursively repeated until there were 9 features remaining. [13] [14]

7.3.1 Optimum Subset of Features

The RFE selected the following 9 columns as the best subset of features for optimising the performances of the regression models:

- lattice_angle_alpha_degree
- lattice_angle_beta_gamma_degree
- lattice_angle_gamma_alpha_degree
- lattice_angle_gamma_degree
- lattice_vector_1_ang
- lattice_vector_2_ang_/_3
- lattice_vector_3_ang
- percent_atom_al
- percent_atom_in

8 Possible extensions or business applications of your project

There are many applications for discovering new transparent conductors. A major application is for touch sensors used in touch screens, which requires transparent conductors. Touch screens are heavily used in products such as smartphones, tablets and display screens. The data analysis carried out in this project would be incredibly useful for technology manufacturers offering products with novel transparent conducting materials.

Another possible application for transparent conductors is in the construction of buildings, resulting in solar energy utilization and energy savings. When using transparent conductors, architects/designers get the added benefit of low infrared emittance. Therefore, it can be used to improve the thermal properties of modern fenestration, depending on whether the TCs are reflecting or not in the near infrared pertinent to solar irradiation. The TCs can serve in "solar control" or "low-emittance" windows. Other applications rely on the electrical conductivity of the TCs, which make them useful as current collectors in solar cells and for inserting and extracting electrical charge in electrochromic "smart windows". It is capable of combining energy efficiency and indoor comfort in buildings.

9 Conclusion

The main goal of this project is to predict the bandgap energy and formation energy for the test dataset using supervised regression models. Firstly, the models have been trained on the dataset after feature engineering and dimensionality reduction techniques had been applied. Feature selection was used in this project to select the optimum subset of features for training the SVR, ridge regression and XGBoost models. The optimum subset was found by removing the redundant or irrelevant columns from the data, leaving the most informative predictors. The initial dataset has 11 features, which was used to create 24 new features. This resulted in a new dataset with 35 features. The new dataset with 35 features requires more memory and processing time, while also causing the regression models used to overfit the training samples. Feature extraction helped alleviate the “curse of dimensionality” problem by combining features. This reduced the number of features required to generalise the data. PCA was also used to explore the amount of variance explained by each column. PCA was vital in the feature engineering process for dimensionality reduction.

The machine learning algorithms: XGBoost, Ridge Regression and SVR was used for prediction. The accuracy of the trained models was evaluated with 10-fold cross validation using RMSLE (Root Mean Squared Logarithmic Error) as the accuracy metric. This was used to assess which model gives the best predictions. As previously discussed, the lower the RMSLE the better the results.

Out of the three implemented models, XGBoost gave the best results with the lowest RMSLE and a reasonable training speed. This was followed by Linear (Ridge) regression and was trained in the shortest time in comparison with the other two models. SVM gives the highest error and takes the longest time to train. Out of the implemented models, an inference can be drawn that boosting algorithms specifically XGBoost is the most suitable for the dataset; as it is designed to push the computation limits to give an accurate result. SVM on the other hand is the least favoured for prediction performance. Although, ridge regression takes the least time, the accuracy of XGBoost precedes the accuracy of Ridge Regression.

To summarise, the Kaggle dataset has been modified to only keep the relevant features and three machine learning models have been trained and tested on this new dataset. Followed by, a comprehensive analysis of all the steps taken in building these models and a critical evaluation of the results to determine the best model to make the predictions. Finally, for the best performing model, the optimum number of principal components to work with on the feature engineered data is highlighted. Followed by, a depiction of the accuracy of the model against the number of principal components.

References

[1] Data mining lecture,
http://www.stat.cmu.edu/ ryantibs/datamining/lectures/16-modr1.pdf

[2] kaggle: Predict Transparent Conductors,
https://www.kaggle.com/c/nomad2018-predict-transparent-conductors

[3] Feature extraction,
https://en.wikipedia.org/wiki/Feature_extraction

[4] Grid Scikit,
http://scikit-learn.org/stable/modules/grid_search.html

[5] Grid search,
https://stackoverflow.com/questions/19335165/cross-validation-and-grid-search

[6] Linear Regression,
http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

[7] Matplotlib Library,
https://matplotlib.org/

[8] Pandas Library,
https://pandas.pydata.org/

[9] Principal Component Analysis
https://en.wikipedia.org/wiki/Principal_component_analysis

[10] Ridge Regression: prodcedures,
https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf

[11] RMSLE,
https://www.quora.com/What-is-the-difference-between-an-RMSE-and-RMSLE-logarithmic-error-and-does-a-high-RMSE-imply-low-RMSLE)

[12] scikit-learn: machine learning in Python,
http://scikit-learn.org/stable/

[13] Scikit: Feature Selection,
http://scikit-learn.org/stable/modules/feature_selection.html

[14] Scikit: Feature Selection,
http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html#sklearn.feature_selection.RFE

[15] Claes G. Granqvist,
http://habana.qfa.uam.es/ lmc/ref/old/granqvist07.pdf
Transparent conductors as solar energy materials: A panoramic review.

[16] SVR: library,
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[17] SVM: svr,
http://www.saedsayad.com/support_vector_machine_reg.htm

[18] Sklearn Linear Model Ridge,
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

[19] XGBoost,
https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

[20] XGBoost: Tree Boosting
http://learningsys.org/papers/LearningSys_2015_paper_32.pdf