Friedrich-Alexander-Universität
Erlangen-Nürnberg

Predicting The Bulk Modulus Of Inorganic Crystals

Data Driven Materials Science

Author: Atharva Sinnarkar

Matriculation Number : 23353068

MSc. Computational Engineering

Module: Materials Informatics

September 24, 2024

Contents

List of Figures

List of Tables

Abstract

The prediction of material properties, such as the bulk modulus, is crucial for advancements in materials science. This report explores the prediction of the bulk modulus of inorganic crystals [1] using machine learning techniques.

Data from Matminer [2], with features generated by the Magpie feature generator, was analyzed using Principal Component Analysis (PCA) to reduce dimensionality. Several regression models, including Linear Regression, Ridge Regression, Lasso Regression, and Tree Regressors, were evaluated based on their coefficient of determination (R^2). Polynomial feature expansion was employed to capture higher-order interactions, enhancing model performance. Feature importance was assessed, identifying the top predictors. Tree-based models, particularly Gradient Boosting, demonstrated superior predictive accuracy over linear models. Kernel Ridge Regression with optimized parameters was also examined. Feature selection methods like LARS and RFE were used to identify key features, providing insights into the factors influencing the bulk modulus. Feature importance analysis identifies key predictors such as density, volume per atom (vpa), and elemental properties. Kernel Ridge Regression with polynomial kernel, especially when data is standardized, shows the highest predictive potential.

The study concludes that Kernel Ridge Regression and Lasso Regression are strong candidates for predictive modeling, balancing accuracy and interpretability.

1 Introduction

In materials science, understanding and predicting material properties are essential for the development of advanced materials with tailored properties. The bulk modulus is a fundamental property characterizing the resistance of a material to volume change under pressure. Accurate prediction of the bulk modulus is crucial for various applications, including designing materials for specific functionalities and optimizing their performance.

Machine learning techniques have shown promise in predicting material properties from composition and structural features. This report aims to explore the prediction of the bulk modulus of inorganic crystals using machine learning methods. The data utilized in this study is sourced from Matminer , a toolkit for materials data mining, with features generated by the Magpie feature generator.

To begin, Principal Component Analysis (PCA) is applied to the dataset to reduce its dimensionality and extract the most relevant features. Subsequently, various regression models, including Linear Regression, Ridge Regression, Lasso Regression, and Tree Regressors, are trained and evaluated for their predictive performance. Polynomial feature expansion is employed to capture potential nonlinear relationships between features, thereby enhancing the models' predictive capabilities.

Furthermore, feature importance analysis is conducted to identify the key predictors influencing the bulk modulus. Both linear and tree-based models are assessed to determine their effectiveness in capturing the complex relationships within the data. Additionally, Kernel Ridge Regression is explored to understand its suitability for predicting material properties.

In addition to model performance evaluation, feature selection methods such as Least Angle Regression (LARS) and Recursive Feature Elimination (RFE) are employed to identify the most important features contributing to the prediction of the bulk modulus. By comparing the results obtained from different models and feature selection techniques, this study aims to provide valuable insights into the factors influencing the bulk modulus of inorganic crystals and guide future materials design and optimization efforts.

2 Background and Theory

2.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a widely used technique in data analysis and machine learning for dimensionality reduction. It transforms the original features into a new set of orthogonal features called principal components (PCs) which capture the most variance in the data. This process helps in simplifying the dataset while retaining its essential patterns.

In the context of predicting the bulk modulus of inorganic crystals, PCA can be particularly useful due to the high dimensionality of the feature space generated by the "Magpie" feature generator from the crystal compositions. By reducing the dimensionality, we can alleviate the curse of dimensionality, improve model performance, and gain insights into the underlying structure of the data.

2.1.1 Steps in PCA

1. **Standardization**: The data is first standardized to have zero mean and unit variance. This ensures that each feature contributes equally to the analysis.

2. **Covariance Matrix Computation**: The covariance matrix of the standardized data is computed to understand how the variables vary together.

3. **Eigenvalue and Eigenvector Calculation**: The eigenvalues and eigenvectors of the covariance matrix are computed. The eigenvectors corresponding to the largest eigenvalues form the principal components.

4. **Principal Components Selection**: The principal components are sorted by the amount of variance they explain, and the top k components are selected to form a reduced feature space. The magnitude of the eigenvalues of the covariance matrix is (proportional to) the amount of explained variance by the corresponding eigenvector (= principal component).

5. **Transformation**: The original data is projected onto the new feature space formed by the selected principal components.

Mathematically, if X is the standardized data matrix, the covariance matrix Σ is given by:

$$\Sigma = \frac{1}{n-1} X^T X$$

The eigenvalue decomposition of Σ gives:

$$\Sigma v = \lambda v$$

where λ are the eigenvalues and v are the eigenvectors. The data is then transformed to:

$$Z = XV$$

where V is the matrix of selected eigenvectors, and Z is the transformed data in the reduced feature space.

2.2 Linear Regression

In machine learning, linear models are fundamental tools used for regression tasks. Specifically, we often employ Linear Least Squares, Lasso, and Ridge regression to model relationships between features and target variables. These methods differ primarily in how they handle regularization, which is crucial for preventing overfitting and improving model generalization.

2.2.1 Linear Least Squares

Linear Least Squares regression seeks to minimize the sum of the squares of the residuals, the differences between observed and predicted values. The model is defined as:

$$\hat{y} = X\beta$$

where $\hat{y}$ is the predicted value, X is the matrix of input features, and β is the vector of coefficients. The objective is to find β that minimizes:

$$\min_{\beta} \|X\beta - y\|_2^2$$

2.2.2 Ridge Regression

Ridge regression (also known as Tikhonov regularization) adds a penalty to the least squares loss function proportional to the square of the magnitude of the coefficients. This helps to shrink the coefficients and handle multicollinearity. The objective function is:

$$\min_{\beta} \left(\|X\beta - y\|_2^2 + \alpha\|\beta\|_2^2\right)$$

where α is the regularization parameter that controls the strength of the penalty. Finding the optimal α is essential for balancing bias and variance.

2.2.3 Lasso Regression

Lasso (Least Absolute Shrinkage and Selection Operator) regression adds a penalty proportional to the absolute value of the magnitude of the coefficients, promoting sparsity in the model by driving some coefficients to zero. The objective function is:

$$\min_{\beta} \left(\|X\beta - y\|_2^2 + \alpha\|\beta\|_1\right)$$

where $\|\beta\|_1$ is the L1 norm of the coefficients, and α is the regularization parameter. Similar to Ridge, finding the optimal α is crucial.

2.2.4 Steps in Optimal Linear Model Task

To identify the optimal linear model for predicting the bulk modulus, we follow these steps:

1. **Data Preprocessing**: Standardize the features to have zero mean and unit variance.
2. **Model Fitting**: Fit Linear Least Squares, Lasso, and Ridge regression models to the data.

3. **Hyperparameter Tuning**: Use GridSearchCV to find the optimal regularization parameter α for both Ridge and Lasso regression.

4. **Feature Importance**: Identify and save the ten most important features based on the largest magnitude of the coefficients.

2.3 Polynomial Expansion

Polynomial feature expansion is a technique used to capture non-linear relationships in data by adding polynomial terms and interaction terms of the original features. This method enhances the capacity of linear models to fit complex patterns without altering their fundamental linear nature. By expanding the feature space, we can fit models that are more expressive and potentially more accurate.

2.3.1 Polynomial Feature Expansion

Polynomial feature expansion involves generating new features from the existing ones by taking powers and interactions of the original features up to a specified degree. For instance, for features x_1 and x_2, a polynomial expansion of degree 2 would include terms such as x_1^2, x_2^2, and $x_1 x_2$.

Mathematically, given a feature vector $X = [x_1, x_2, \ldots, x_p]$, the polynomial expansion of degree d includes all terms of the form:

$$x_1^{a_1} x_2^{a_2} \ldots x_p^{a_p}$$

where $a_1 + a_2 + \ldots + a_p \leq d$. The expanded feature space allows linear models to learn higher-order interactions between features.

2.3.2 Choice of Polynomial Degree

The choice of the polynomial degree d is a trade-off between model complexity and computational expense. Higher degrees can capture more complex relationships but also increase the risk of overfitting and the computational cost. A common approach is to start with a low degree and gradually increase it while monitoring model performance and computational feasibility.

2.3.3 Steps in Polynomial Expansion Task

To incorporate polynomial features and train the linear models, we proceed as follows:

1. **Data Preprocessing**: Standardize the original features to have zero mean and unit variance.

2. **Polynomial Expansion**: Generate polynomial features up to a chosen degree d.

3. **Model Fitting**: Train Linear Least Squares, Ridge, and Lasso regression models on the expanded feature set.

4. **Hyperparameter Tuning**: Use the optimal regularization parameters for Ridge and Lasso found in the previous task.

5. **Feature Importance**: Identify and save the ten most important polynomial features based on the largest magnitude of the coefficients. For this, all the nonlinear features (the interaction) also need to be standardized.

2.4 Tree Regressors

In the realm of machine learning, decision tree regressors are popular models for their simplicity and interpretability. Decision trees split the data into subsets based on the values of the input features, creating a tree-like model of decisions. However, decision trees can suffer from overfitting, especially if the tree is too deep. To mitigate this, we can tune the depth of the tree and employ ensemble methods such as AdaBoost, Gradient Boosting, and Histogram-Based Boosting to enhance performance.

2.4.1 Decision Tree Regressor

A decision tree regressor models the target variable by learning simple decision rules inferred from the features. The goal is to split the data into subsets such that the target values within each subset are as homogeneous as possible. The splits are based on feature values and are chosen to minimize a loss function, typically the mean squared error (MSE) for regression tasks.

The depth of the tree, which is the number of splits from the root to the deepest leaf, is a critical hyperparameter. A deeper tree can model more complex relationships but may overfit the training data, leading to poor generalization on unseen data.

2.4.2 Feature Importance in Decision Trees

Feature importance in decision trees is determined by the amount that each feature contributes to decreasing the impurity (viz, variance for regression tasks) across all the nodes where the feature is used to split the data. Impurity is a measure used in decision trees to evaluate the quality of a split at each node. It quantifies how mixed the target values are within a node. The goal of a decision tree is to create nodes that have low impurity, meaning that the target values in each node are as homogeneous as possible. The more a feature reduces the impurity, the higher its importance.

Mathematically, the importance I_j of feature j is computed as:

$$I_j = \sum_{t \in T} \Delta I_t$$

where T is the set of all nodes, and ΔI_t is the decrease in impurity at node t that results from splitting on feature j.

2.4.3 AdaBoost Regressor

AdaBoost (Adaptive Boosting) is an ensemble method that combines the predictions of multiple base estimators (like decision trees) to improve robustness and accuracy. It builds the model iteratively, adjusting the weights of training samples based on the errors of previous models. Misclassified points receive higher weights so that subsequent models focus more on these hard-to-predict samples.

The final model is a weighted sum of the individual models:

$$F(x) = \sum_{m=1}^{M} \alpha_m h_m(x)$$

where $h_m(x)$ is the m-th base model and α_m is the weight assigned to that model based on its accuracy.

2.4.4 Steps in Tree Regressor Task

To train decision tree regressors and enhance their performance using AdaBoost, we follow these steps:

1. **Data Preprocessing**: Split the data into training and testing sets.
2. **Model Tuning**: Perform a grid search to find the optimal tree depth for the decision tree regressor.
3. **Model Training**: Train the decision tree regressor with the optimal depth on the training data.
4. **Boosting**: Apply the AdaBoost method to the decision tree regressor to improve performance.
5. **Feature Importance**: Extract and save the ten most important features based on their importance scores from the AdaBoost model.

2.5 Kernel Ridge Regression

Kernel Ridge Regression (KRR) is a powerful regression technique that combines ridge regression with the kernel trick. This allows KRR to model non-linear relationships between the features and the target variable without explicitly transforming the input data into higher-dimensional space.

2.5.1 Kernel Ridge Regression

Ridge regression is a linear model that includes a regularization term to prevent overfitting. The objective function is:

$$\min_{\beta} \left(\|X\beta - y\|_2^2 + \alpha \|\beta\|_2^2 \right)$$

where α is the regularization parameter.

Kernel Ridge Regression extends this concept by applying the kernel trick, which implicitly maps the input features into a high-dimensional feature space using a kernel function. The objective function becomes:

$$\min_{\beta} \left(\|K\beta - y\|_2^2 + \alpha \|\beta\|_2^2 \right)$$

where K is the kernel matrix computed as $K_{ij} = k(x_i, x_j)$, and $k(\cdot, \cdot)$ is the kernel function.

2.5.2 Kernel Functions

Two commonly used kernels are:

Radial Basis Function (RBF) Kernel:

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

where σ is the kernel parameter controlling the width of the Gaussian.

Polynomial Kernel:

$$k(x_i, x_j) = (x_i \cdot x_j + c)^d$$

where c is a constant term and d is the degree of the polynomial.

2.5.3 Suitability for Feature Selection

Kernel Ridge Regression, especially with non-linear kernels like RBF and polynomial, tends to model complex interactions between features. While this is beneficial for prediction tasks, it makes feature selection challenging. The kernel function maps the input features to a high-dimensional space where the original feature contributions become entangled, complicating the interpretation of feature importance. Unlike linear models where coefficients directly indicate feature importance, non-linear kernels distribute feature importance across the transformed feature space, making it difficult to isolate the impact of individual features. Consequently, Kernel Ridge Regression, except when using a linear kernel, is not particularly suited for feature selection in scenarios where interpretability and simplicity are prioritized.

2.5.4 Steps in Kernel Ridge Regression Task

To implement Kernel Ridge Regression with two different kernels and optimize the parameters, we follow these steps:

1. **Data Preprocessing**: Standardize the features to have zero mean and unit variance.
2. **Kernel Selection**: Choose two kernels, e.g., RBF and Polynomial.
3. **Parameter Optimization**: Perform grid search to find the optimal kernel parameters and regularization parameter α.
4. **Model Training**: Train the KRR models with the optimal parameters.
5. **Feature Importance**: Discuss why KRR with non-linear kernels is not suitable for feature selection.

2.6 Compare performance across different models

In machine learning, comparing the performance of different models is crucial to understand their strengths and weaknesses and to select the best model for a given task. This involves evaluating models on standardized and non-standardized data, adjusting the training set size, and examining how these factors affect performance and feature importance.

2.6.1 Standardization

Standardization is a preprocessing technique where features are rescaled to have zero mean and unit variance. This is especially important for models that rely on the magnitude of features, such as Ridge Regression and Lasso, as it ensures that all features contribute equally to the model. Non-standardized data can lead to biased results, where features with larger scales dominate the model.

2.6.2 Model Performance Metrics

To evaluate model performance, regression metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) are commonly used:

- **Mean Squared Error (MSE)**: Measures the average of the squares of the errors, providing a sense of how close the predicted values are to the actual values.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error (RMSE)**: The square root of MSE, giving the error in the same units as the target variable.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

- **R-squared (R^2)**: Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

2.6.3 Impact of Training Set Size

The size of the training set significantly impacts model performance. Generally, increasing the training set size improves the model's ability to generalize by providing more data to learn from. However, it also increases computational cost and training time. Evaluating models with different training set sizes helps in understanding the trade-off between model performance and computational efficiency.

2.6.4 Steps in 'Compare performance across different models' Task

To compare model performance across different configurations, we follow these steps:

1. **Data Preprocessing**: Standardize the features, including polynomial features.
2. **Model Training**: Train each model on both standardized and non-standardized data.
3. **Varying Training Set Size**: Adjust the training set size and evaluate model performance.
4. **Performance Evaluation**: Use regression metrics such as MSE, RMSE, and R^2 to compare performance.
5. **Feature Importance Analysis**: Examine how standardization and training set size affect the importance of features.

2.7 Feature Selection

Feature Selection:
In this task, we will compare the most important features identified previously across different models and with methods specifically designed for feature selection.

Least Angle Regression: Least Angle Regression (LARS) is a regression algorithm that is particularly efficient for high-dimensional data, where the number of predictors p is much larger than the number of observations n. LARS is closely related to forward stepwise regression and the LASSO (Least Absolute Shrinkage and Selection Operator). LARS operates as follows:

1. Start with all coefficients $\beta = 0$.
2. Identify the predictor most correlated with the response. Move the coefficient of this predictor from 0 towards its least squares value (i.e., the direction of the steepest descent).
3. Continue until another predictor has as much correlation with the current residual. Then move the coefficients of these two predictors in a way that keeps their correlations equal.
4. Continue this process until all predictors are included in the model.

When combined with LASSO, the LARS algorithm modifies the coefficient update steps to perform soft thresholding, which can set some coefficients exactly to zero, thus performing variable selection.

Recursive Feature Elimination (RFE): Recursive Feature Elimination (RFE) is a feature selection technique that aims to select features by recursively considering smaller and smaller sets of features. The basic idea of RFE is:

1. Train the model on the initial set of features.
2. Rank the features based on their importance (e.g., weights in a linear model, or feature importance in a tree-based model).
3. Remove the least important features.
4. Repeat the process with the reduced set until the desired number of features is reached.

This method ensures that the most relevant features are retained by continuously eliminating the less significant ones, allowing the model to focus on the most informative subset of features.

Comparison: To perform the comparison:

- Analyze the overlap between the sets of features selected by different methods (e.g., LARS with LASSO, RFE with tree model, RFE with linear model).
- Evaluate the physical interpretability of the selected features. Determine if the features make sense in the context of the problem domain.

- Investigate the stability of feature selection across different training set sizes by varying the size of the training set and observing the consistency of the selected features.

This comprehensive comparison will help in understanding the robustness and reliability of the feature selection methods, as well as their practical relevance to the problem at hand.

The selected features are evaluated for their physical interpretability based on their relevance to material properties. For example, features like magnetic properties, electronegativity, and density are physically motivated as they are crucial in determining material behavior. If selected features are judged as not physically interpretable, it implies that the model might be relying on spurious correlations rather than meaningful physical relationships, leading to potentially unreliable predictions. The report confirms the physical relevance of features such as melting points, column numbers, and volume per atom, ensuring they make sense in the context of the problem domain

3 Methods

3.1 Implementation of PCA

We implemented Principal Component Analysis (PCA) using Python as per the script provided in the assignment. The implementation is encapsulated within the class `PrincipalComponentAnalysis`, which includes methods to perform training (`train`), transformation (`transform`), and back transformation (`backtransform`) on the dataset. The PCA implementation follows these steps:

1. **Initialization**: The class `PrincipalComponentAnalysis` is initialized with the number of components `n_components` to retain.

```
class PrincipalComponentAnalysis():
    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.eigenvalues = None
        self.eigenvectors = None
        self.means = None
```

2. **Training**: The `train` method computes the principal components based on the input data `xtrain`.

```
def train(self, xtrain):
    self.means = np.mean(xtrain, axis=0)
    xtrain_centered = xtrain - self.means
    covariance = np.cov(xtrain_centered.T)
    self.eigenvalues, self.eigenvectors = np.linalg.eigh(covariance)
    idxs = np.argsort(self.eigenvalues)[::-1]
    self.eigenvalues = self.eigenvalues[idxs]
    self.eigenvectors = self.eigenvectors[:, idxs]
    self.components = self.eigenvectors[:, :self.n_components]
    return self.eigenvalues, self.eigenvectors
```

3. **Transformation**: The `transform` method projects the input data onto the principal components.

```
def transform(self, xtrain):
    xtrain_centered = xtrain - self.means
    return np.dot(xtrain_centered, self.components)
```

4. **Back Transformation**: The `backtransform` method reconstructs the data from its principal component representation back to the original space.

```
def backtransform(self, x_transformed):
    return np.dot(x_transformed, self.components.T) + self.means
```

The implementation was validated using the provided StudOn program to ensure correctness.

3.1.1 Spectrum of Principal Components

To analyze the data, we performed PCA on the feature set and plotted the cumulative spectrum of the principal components. The script `pca_plot_spectrum.py` was used for this purpose. The steps are as follows:

1. **Data Loading**: The feature dataset was loaded from a CSV file.

```
X = read_csv('features-bulk.csv')
```

2. **PCA Training**: PCA was performed on the dataset using the `PrincipalComponentAnalysis` class.

```
pca = PrincipalComponentAnalysis(X.shape[1])
eigenvalues, _ = pca.train(X)
```

3. **Variance Calculation**: The explained variance ratio for each principal component was computed and the cumulative variance ratio was calculated.

```
explained_variance_ratio = eigenvalues / np.sum(eigenvalues)
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)
```

4. **Plotting**: Two plots were generated:
   - The first plot shows the contribution of each principal component to the total variance.
   - The second plot shows the cumulative explained variance ratio as a function of the number of principal components.

3.2 Model Training and Evaluation

The task involves finding the optimal model for making predictions on provided data, measuring prediction performance using the coefficient of determination R^2. The tasks are divided into several subtasks:

3.2.1 The Optimal Linear Model

The objective is to fit Linear Least Squares, Lasso, and Ridge models to the data and identify the optimal regularization parameters for Lasso and Ridge using grid search. The implementation proceeds as follows:

1. Data Loading and Preprocessing First, we load the feature and target datasets from CSV files and standardize the feature dataset.

1. **Data Loading**: The feature and target datasets are loaded using `pandas`.

```
features_df = pd.read_csv("features-bulk.csv")
target_df = pd.read_csv("target-bulk.csv")

# Extract features and target
X = features_df.values
y = target_df.values.ravel()
print("size of feature dataset is:", X.shape)
print("size of target dataset is:", y.shape)
```

2. **Standardization**: The features are standardized to have zero mean and unit variance.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("size of scaled feature dataset is:", X_scaled.shape)
```

2. Model Fitting and Grid Search We fit a Linear Regression model and perform grid search to find the optimal regularization parameters for Lasso and Ridge.

1. **Linear Regression**: Fit a Linear Least Squares model to the scaled data.

```
linear_model = LinearRegression()
linear_model.fit(X_scaled, y)
```

2. **Grid Search for Lasso**: Perform grid search with cross-validation to find the optimal α for Lasso.

```
lasso_params = {'alpha': [0.01, 0.02, 0.1, 1, 10]}
lasso = Lasso(max_iter=38000)
lasso_grid = GridSearchCV(lasso, param_grid=lasso_params, cv=5)
lasso_grid.fit(X_scaled, y)
df_lasso = pd.DataFrame(lasso_grid.cv_results_)
print("Cross Validation Table For Lasso:\n", df_lasso)
lasso_best_alpha = lasso_grid.best_params_['alpha']
lasso_best = lasso_grid.best_estimator_
```

3. **Grid Search for Ridge**: Perform grid search with cross-validation to find the optimal α for Ridge.

```
ridge_params = {'alpha': [0.01, 0.1, 1, 2, 10]}
ridge = Ridge(max_iter=38000)
ridge_grid = GridSearchCV(ridge, param_grid=ridge_params, cv=5)
ridge_grid.fit(X_scaled, y)
df_ridge = pd.DataFrame(ridge_grid.cv_results_)
print("Cross Validation Table For Ridge:\n", df_ridge)
ridge_best_alpha = ridge_grid.best_params_['alpha']
ridge_best = ridge_grid.best_estimator_
```

3. Feature Selection We extract and save the ten most important features (with the largest norm of weights) for each model.

1. **Linear Regression Features**: Extract the top 10 features using `SelectFromModel`.

```
linear_selector = SelectFromModel(linear_model, max_features=10)
linear_selector.fit(X_scaled, y)
linear_selected_features = features_df.columns[linear_selector.get_support()].
    ↪ tolist()
```

2. **Lasso Features**: Extract the top 10 features using `SelectFromModel`.

```
lasso_selector = SelectFromModel(lasso_best, max_features=10)
lasso_selector.fit(X_scaled, y)
lasso_selected_features = features_df.columns[lasso_selector.get_support()].tolist
    ↪ ()
```

3. **Ridge Features**: Extract the top 10 features using `SelectFromModel`.

```
ridge_selector = SelectFromModel(ridge_best, max_features=10)
ridge_selector.fit(X_scaled, y)
ridge_selected_features = features_df.columns[ridge_selector.get_support()].tolist
    ↪ ()
```

4. Visualization of Model Predictions Finally, we visualize the predictions of the three models by plotting the predicted target values against the actual target values.

1. **Predictions**: Generate predictions for the scaled dataset using each model.

```
linear_predictions = linear_model.predict(X_scaled)
lasso_predictions = lasso_best.predict(X_scaled)
ridge_predictions = ridge_best.predict(X_scaled)
```

2. **Plotting**: Create scatter plots to compare predicted and actual target values.

3.2.2 Polynomial Expansion

The task involves performing polynomial feature expansion with cross terms, up to a specified order, followed by training the previously mentioned linear models—Least Squares, Ridge, and Lasso—with the expanded features. Additionally, the goal is to save the ten most important features/terms, determined by the largest norm of weights, for later use.

1. Polynomial Feature Expansion We begin by defining the order of polynomial expansion, denoted as *poly_order*. This parameter specifies the degree of polynomial features to be generated, such as quadratic or higher-order features.

Perform polynomial feature expansion on the standardized feature dataset (X_{scaled}) using `PolynomialFeatures` from scikit-learn. The degree of the polynomial expansion is set to *poly_order*. The resulting feature matrix is denoted as X_{poly}.

```
poly_order = 2
poly = PolynomialFeatures(degree=poly_order, include_bias=False)
X_poly = poly.fit_transform(X_scaled)
print("Size of polynomial feature dataset is:", X_poly.shape)
```

2. Model Training and Feature Selection After polynomial expansion, the three linear models—Linear Regression, Lasso, and Ridge—are trained with the expanded feature dataset. Subsequently, the top ten most important features/terms are extracted and saved for each model.

1. **Linear Regression with Polynomial Features**: Fit a Linear Regression model to the polynomial feature dataset (X_{poly}).

   ```
   linear_model_poly = LinearRegression()
   linear_model_poly.fit(X_poly, y)
   ```

   Extract and save the top ten features/terms selected by Linear Regression with polynomial expansion.

   ```
   linear_coef_abs = np.abs(linear_model_poly.coef_)
   top_linear_indices = np.argsort(linear_coef_abs)[-10:][::-1]
   top_linear_features = [poly.get_feature_names_out(input_features=features_df.
       ↪ columns)[i] for i in top_linear_indices]
   print("Top 10 features/terms selected by Linear Regression with polynomial
       ↪ expansion:", top_linear_features)
   ```

2. **Lasso with Polynomial Features**: Fit a Lasso model with the optimal regularization parameter obtained earlier to the polynomial feature dataset (X_{poly}).

   ```
   lasso_poly = Lasso(alpha=lasso_best_alpha, max_iter=38000)
   lasso_poly.fit(X_poly, y)
   ```

   Extract and save the top ten features/terms selected by Lasso with polynomial expansion.

   ```
   lasso_coef_abs = np.abs(lasso_poly.coef_)
   top_lasso_indices = np.argsort(lasso_coef_abs)[-10:][::-1]
   top_lasso_features = [poly.get_feature_names_out(input_features=features_df.columns
       ↪ )[i] for i in top_lasso_indices]
   print("Top 10 features/terms selected by Lasso with polynomial expansion:",
       ↪ top_lasso_features)
   ```

3. **Ridge with Polynomial Features**: Fit a Ridge model with the optimal regularization parameter obtained earlier to the polynomial feature dataset (X_{poly}).

```
ridge_poly = Ridge(alpha=ridge_best_alpha, max_iter=38000)
ridge_poly.fit(X_poly, y)
```

Extract and save the top ten features/terms selected by Ridge with polynomial expansion.

```
ridge_coef_abs = np.abs(ridge_poly.coef_)
top_ridge_indices = np.argsort(ridge_coef_abs)[-10:][::-1]
top_ridge_features = [poly.get_feature_names_out(input_features=features_df.columns
    ↪ )[i] for i in top_ridge_indices]
print("Top 10 features/terms selected by Ridge with polynomial expansion:",
    ↪ top_ridge_features)
```

3.2.3 Tree Regressors

To implement tree regressors, we start by splitting the dataset into training and testing sets. This split is crucial to assess the performance of the models accurately.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.4,
    ↪ random_state=42)
```

Next, we conduct a grid search to find the optimal depth for the decision tree regressor. This process involves testing various depths and selecting the one with the best performance.

```
# Define the parameter grid for grid search on decision tree depth
tree_params = {'max_depth': [1, 3, 5, 7, 10, 15]}

# Perform grid search for decision tree regressor
tree_reg = DecisionTreeRegressor(random_state=42)
tree_grid = GridSearchCV(tree_reg, param_grid=tree_params, cv=5)
tree_grid.fit(X_train, y_train)
df_tree = pd.DataFrame(tree_grid.cv_results_)
print("Cross Validation Table For Decision Tree:\n", df_tree)
tree_best_depth = tree_grid.best_params_['max_depth']
tree_best = tree_grid.best_estimator_
```

Following the determination of the optimal tree depth, we fit the selected modification method to the decision tree regressor. For this demonstration, we choose the Adaboost modification method. Other options include Gradient-Boost and Hist-Boost.

```
# Fit the selected modification method to the decision tree regressor
base_tree = DecisionTreeRegressor(max_depth=tree_best_depth)
boosting_method = AdaBoostRegressor(random_state=42)
boosting_method.base_estimator_ = base_tree
boosting_method.fit(X_train, y_train)
```

Finally, we extract and save the ten most important features for the selected modification method. These features provide insights into the factors that significantly influence the model's predictions.

```
# Extract and save the ten most important features for the selected modification method
boosting_features_importance = boosting_method.feature_importances_
top_boosting_indices = np.argsort(boosting_features_importance)[-10:][::-1]
```

```
top_boosting_features = features_df.columns[top_boosting_indices].tolist()
print("Top 10 features selected by the modification method:", top_boosting_features)
```

3.2.4 Kernel Ridge Regression

Kernel Ridge Regression is employed with the aim of optimizing kernel parameters and the regularization parameter through grid search. However, it's crucial to note that Kernel Ridge Regression, except with the linear kernel, is not particularly suited for feature selection. We will discuss this limitation after presenting the implementation details.

Parameter Grids and Grid Search For Kernel Ridge Regression, we define a parameter grid encompassing various combinations of hyperparameters. These hyperparameters include the regularization parameter (alpha), the type of kernel (linear, radial basis function (RBF), and polynomial), the kernel coefficient (gamma), and the degree of the polynomial kernel.

```
# Define parameter grids for grid search
kernel_ridge_params = {'alpha': [0.01, 0.1, 1, 10],
                       'kernel': ['linear', 'rbf', 'poly'],
                       'gamma': [0.01, 0.1, 1, 10],
                       'degree': [2, 3, 4]}
```

The grid search is then performed using cross-validation to identify the optimal combination of hyperparameters that yields the best performance on the given dataset.

```
# Perform grid search for Kernel Ridge Regression
kernel_ridge = KernelRidge()
kernel_ridge_grid = GridSearchCV(kernel_ridge, param_grid=kernel_ridge_params, cv=5)
kernel_ridge_grid.fit(X_scaled, y)
df_kernel_ridge = pd.DataFrame(kernel_ridge_grid.cv_results_)
print("Cross Validation Table For Kernel Ridge Regression:\n", df_kernel_ridge)

# Extract best parameters
kernel_ridge_best_params = kernel_ridge_grid.best_params_
print("Best parameters for Kernel Ridge Regression:", kernel_ridge_best_params)
print("")
```

The best parameters obtained through the grid search are then extracted and printed for further analysis.

3.2.5 Compare Performance Across Different Models

In this task, we aim to compare the performance of various regression models across different scenarios, including standardized and non-standardized data, as well as different sizes of the training set. We will discuss how both standardization and training set size affect model performance and the emergence of the most important features.

1. Evaluation Function First, we define a function to evaluate the performance of each model using a regression metric of choice, in this case, the coefficient of determination (R2 score).

```
# Define a function to evaluate model performance
def evaluate_model(model, X_train, X_test, y_train, y_test):
    # Fit the model
    model.fit(X_train, y_train)
```

```
    # Make predictions
    predictions = model.predict(X_test)
    # Calculate R2 score
    ss_res = np.sum((y_test - predictions) ** 2)
    ss_tot = np.sum((y_test - np.mean(y_test)) ** 2)
    r2 = 1 - (ss_res / ss_tot)
    return r2
```

2. Model Selection We select several regression models to compare, including Linear Regression, Lasso Regression, Ridge Regression, Decision Tree, Adaboost, and Kernel Ridge Regression.

```
# Define different models
models = {
    "Linear Regression": LinearRegression(),
    "Lasso Regression": Lasso(alpha=lasso_best_alpha, max_iter=1000000),
    "Ridge Regression": Ridge(alpha=ridge_best_alpha, max_iter=38000),
    "Decision Tree": DecisionTreeRegressor(max_depth=tree_best_depth, random_state=42),
    "Adaboost": AdaBoostRegressor(random_state=42),
    "Kernel Ridge Regression": KernelRidge(alpha=kernel_ridge_best_params['alpha'],
                                           kernel=kernel_ridge_best_params['kernel'],
                                           gamma=kernel_ridge_best_params['gamma'],
                                           degree=kernel_ridge_best_params['degree'])
}
```

3. Varying Training Set Size We define a range of training set sizes to test the models' performance. We will split the data into training and testing sets with each specified size and evaluate model performance accordingly.

```
# Define training set sizes to test
train_sizes = [0.9, 0.7, 0.6, 0.4, 0.3, 0.2, 0.1]
```

4. Model Evaluation We iterate over each model and training set size, evaluating model performance on both standardized and non-standardized data. This allows us to observe how standardization and training set size impact model performance.

```
# Loop over models and training set sizes
for model_name, model in models.items():
    print("Model:", model_name)
    for train_size in train_sizes:
        # Split the data into training and testing sets with the specified size
        X_train_std, X_test_std, y_train_std, y_test_std = train_test_split(X_scaled, y,
    ↪ test_size=(1-train_size), random_state=42)
        X_train_nonstd, X_test_nonstd, y_train_nonstd, y_test_nonstd = train_test_split(X
    ↪ , y, test_size=(1-train_size), random_state=42)

        # Evaluate model performance on standardized data
        r2_std = evaluate_model(model, X_train_std, X_test_std, y_train_std, y_test_std)

        # Evaluate model performance on non-standardized data
        r2_nonstd = evaluate_model(model, X_train_nonstd, X_test_nonstd, y_train_nonstd,
    ↪ y_test_nonstd)

        print(f"Train Size: {train_size}, Standardized Data R2: {r2_std}, Non-
    ↪ Standardized Data R2: {r2_nonstd}")
```

3.3 Feature Selection

In this task, we compare the most important features obtained across various models and methods specifically designed for feature selection.

3.3.1 Task 3.1: Least Angle Regression

Least Angle Regression (LARS) with LASSO regularization is applied to find the ten most important features. LARS is a regression algorithm that finds the best fit to the data by sequentially adding predictors. It does this in a 'forward stepwise' manner, incrementally updating the coefficient estimates to minimize the residual sum of squares. In the case of LASSO, it adds a penalty term to the coefficient estimates, effectively shrinking some coefficients to zero, thus performing feature selection.

```
# Load data
features_df = pd.read_csv("features-bulk.csv")
target_df = pd.read_csv("target-bulk.csv")

# Convert dataframes to numpy arrays
X = features_df.values
y = target_df.values.ravel()

# Apply LARS with LASSO regularization
_, _, coefs = lars_path(X, y, method='lasso', verbose=True)

# Get the most important features
important_features_indices = np.argsort(np.abs(coefs.sum(axis=1)))[-10:]
important_features = features_df.columns[important_features_indices]
print("Top 10 most important features:")
print(", ".join(important_features))

# Perform polynomial + interaction expansion
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# Apply LARS with LASSO regularization on expanded features
_, _, coefs_poly = lars_path(X_poly, y, method='lasso', verbose=True)

# Get the most important features from expanded features
important_features_indices_poly = np.argsort(np.abs(coefs_poly.sum(axis=1)))[-10:]
important_features_poly = poly.get_feature_names_out(features_df.columns)[
    ↪ important_features_indices_poly]
print("\nTop 10 most important features after polynomial expansion:")
print(", ".join(important_features_poly.tolist()))
```

3.3.2 Task 3.2: Recursive Feature Elimination

Recursive Feature Elimination (RFE) is applied with both a tree model and a linear model (LASSO) to find the ten most important features. RFE is a feature selection technique that works by recursively removing features, fitting the model, and evaluating the performance until the desired number of features is reached.

```
# Define models
tree_model = DecisionTreeRegressor()
linear_model = Lasso(max_iter=10000)  # Set max_iter to avoid convergence warning
```

```
# RFE with tree model
rfe_tree = RFE(tree_model, n_features_to_select=10)
rfe_tree.fit(X, y)
important_features_tree = features_df.columns[rfe_tree.support_]

print("Top 10 most important features using Recursive Feature Elimination with a tree
    ↪ model:")
print(", ".join(important_features_tree))

# Standardize features for linear model
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# RFE with linear model (LASSO)
rfe_linear = RFE(linear_model, n_features_to_select=10)
rfe_linear.fit(X_scaled, y)
important_features_linear = features_df.columns[rfe_linear.support_]

print("Top 10 most important features using Recursive Feature Elimination with a linear
    ↪ model (LASSO):")
print(", ".join(important_features_linear))
```

This code snippet illustrates the application of RFE with both a tree model and a linear model to identify the ten most important features in the dataset.

4 Discussion and Results

4.1 PCA Results

The PCA analysis revealed that a small number of principal components could explain a significant portion of the variance in the data. This implies that the data is well-suited for dimensionality reduction. By identifying the directions (principal components) that capture the most variance, PCA allowed us to reduce the dimensionality of the dataset while preserving most of the information. This reduction simplified the complexity of the data, making it easier to visualize and analyze, and lead to more efficient computation in subsequent analyses.

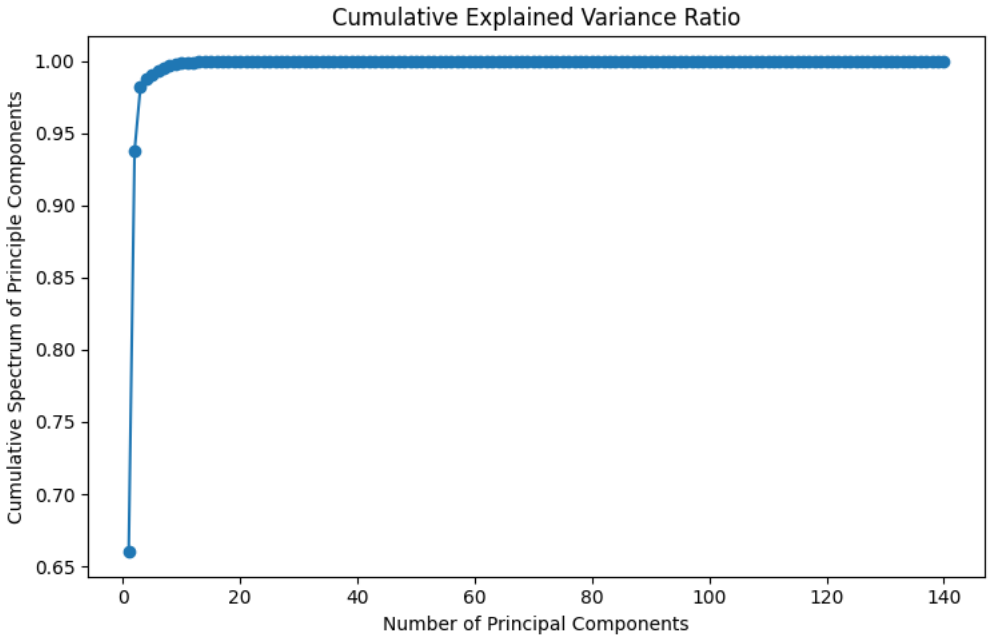

Figure 1: Cumulative Spectrum

Figure 1 shows the cumulative spectrum of principal components.

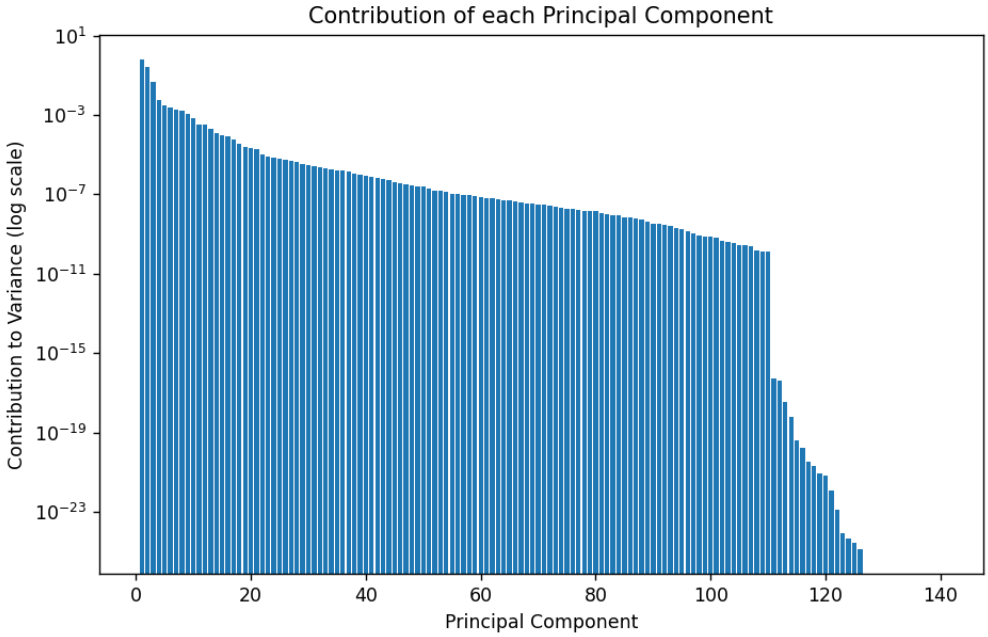

Figure 2: Contribution to Variance

Figure 2 shows the contribution to variance by each principal components.

The results show that:

- Most of the variance in the data is captured by the first few principal components. This suggests that the original dataset has a lot of redundancy.
- The cumulative explained variance ratio quickly levels off after a few principal components. This suggests that we can reduce the dimensionality of the data without losing much information.

- The cumulative explained variance ratio plot shows that around 2 principal components explain 95% of the variance, suggesting 2 dimensions are enough to capture most of the data. If we want very detailed examination, we can plot most of the data in 10-12 dimensions as per Figure 2.

4.1.1 Using the first few PCAs as features for regression and testing the performance for linear ridge regression and some-kernel ridge regression.

Using the first few PCA components as features for regression can be highly beneficial. PCA helps in reducing the dimensionality of the data while retaining most of the variance, which simplifies the regression models and can improve their performance by removing noise and collinearity.
Implementation and Results: In the implementation, the first 10 principal components were used for regression. The data was standardized before applying PCA to ensure that all features contributed equally. The models were then evaluated using both linear ridge regression and kernel ridge regression..
Linear Ridge Regression:

- MSE (Mean Squared Error): 1344.731852348066
- R^2 (Coefficient of Determination): 0.723211429504639

Kernel Ridge Regression:

- MSE (Mean Squared Error): 473.8469550262295
- R^2 (Coefficient of Determination): 0.902467230856266

Graph: The plot of actual vs. predicted values for both linear ridge regression and kernel ridge regression shows the comparative performance of the two models. The kernel ridge regression indicates a better fit compared to linear ridge regression, with predictions closely following the line of perfect prediction.

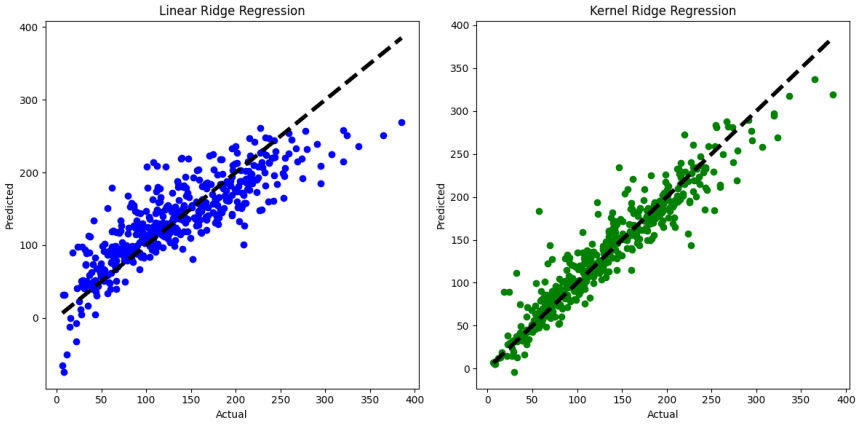

Figure 3: Actual vs. Predicted Values for Linear and Kernel Ridge Regression

Analysis: The kernel ridge regression significantly outperforms linear ridge regression in terms of both MSE and R^2, indicating that it captures the underlying data patterns more effectively. Using PCA components as features allows both models to handle the reduced dimensionality data efficiently, resulting in improved performance metrics compared to potentially using the original high-dimensional data. The scatter plots clearly

show that kernel ridge regression provides a tighter fit around the line of perfect prediction, suggesting better generalization and predictive power..
Conclusion: Using the first few PCA components as features for regression is a valid and effective approach, particularly with kernel ridge regression, which shows superior performance in this context. The results highlight the importance of dimensionality reduction techniques like PCA in enhancing the predictive capabilities of regression models. Also, the extrapolation can be identified by comparing the range of the training data with the predictions. If the model predicts values outside the range of the training data, it is extrapolating. In our case, it is not predicting any values beyond the training data set.

4.2 Finding the Optimal Model

In this section, we aim to identify the model that yields optimal predictions for our dataset. We evaluate the prediction performance using the coefficient of determination R^2. Before proceeding, it is essential to thoroughly understand each sub-task to ensure effective implementation.
The size of the feature dataset is $(1181, 140)$, and the size of the target dataset is (1181). After scaling, the size of the feature dataset remains unchanged at $(1181, 140)$. This indicates that there are 1181 data points and each data point has 140 features. For the target dataset, it only contains 1181 rows, without any additional dimension for the number of features, because each data point has a single target value. Therefore, the target set is represented as a one-dimensional array of 1181 values instead of a two-dimensional array with dimensions (1181, 1)

4.2.1 The Optimal Linear Model

Linear Regression: The blue points are closely clustered around the black dashed line (the ideal scenario where predicted and actual values match perfectly), indicating a good fit. However, there is some noticeable deviation from the line at higher actual values, suggesting that the model may struggle with extrapolating to very high values.

Lasso Regression: The red points also show a good fit, with a similar pattern to the Linear Regression model. The deviation from the line is less pronounced than for Linear Regression, suggesting that Lasso might be slightly better at handling outliers and extreme values.

Ridge Regression: The green points show a good fit as well, but the scatter is slightly more spread out compared to the other two models. This suggests that Ridge Regression might be slightly less accurate than the other two models, but still performs reasonably well.

Observations: Below are the tables showing Top 10 features selected by Linear, Lasso and Ridge Regressions.

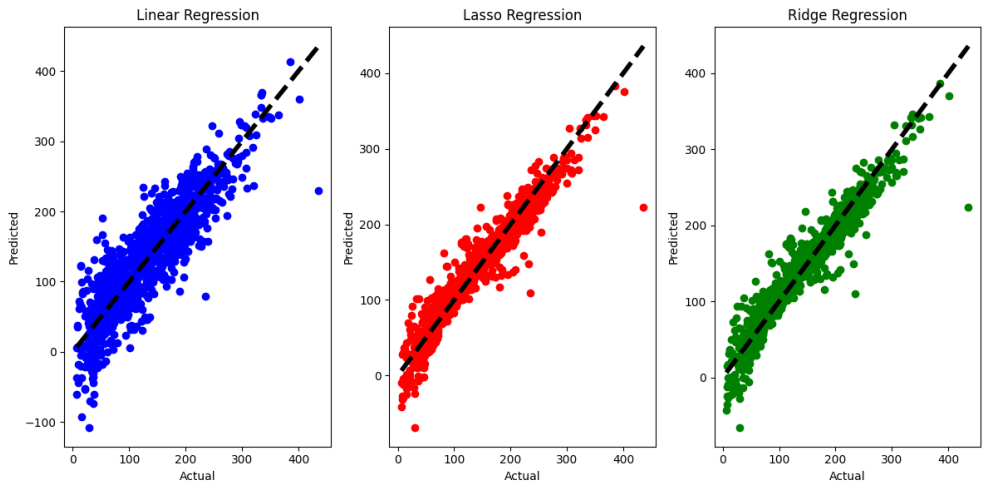

Figure 4: Predicted vs. Actual Values for Linear Regression, Lasso Regression, and Ridge Regression

Linear Regression	**Lasso**
MagpieData minimum Number	MagpieData mean Number
MagpieData maximum Number	MagpieData mean AtomicWeight
MagpieData range Number	MagpieData mean MeltingT
MagpieData minimum MendeleevNumber	MagpieData mean Row
MagpieData range MendeleevNumber	MagpieData mean CovalentRadius
MagpieData minimum NsValence	MagpieData mean NpValence
MagpieData range NsValence	MagpieData mean GSvolume_pa
Minimum oxidation state	MagpieData mean GSmagmom
Maximum oxidation state	Density
Range oxidation state	Packing fraction

Table 1: Top 10 features selected by Linear and Lasso Regression

Ridge
MagpieData minimum Number
MagpieData maximum Number
MagpieData mean Number
MagpieData mode Number
MagpieData mode AtomicWeight
MagpieData mean MeltingT
MagpieData mean Row
MagpieData mean CovalentRadius
MagpieData mean GSvolume_pa
Density

Table 2: Top 10 features selected by Ridge

Analysis of Cross-Validation Scores for Lasso and Ridge Regression Model

Lasso Regression

- Trend: The figure 5 shows a clear decreasing trend in the mean test score as alpha increases. This is expected behavior for Lasso.
- High Alpha: When alpha is large, Lasso penalizes the coefficients heavily, leading to simpler models with more coefficients set to zero. This can result in underfitting if the model is too simplified, hence the lower score.
- Low Alpha: As alpha decreases, Lasso allows for more coefficients to be non-zero, leading to a more complex model. This can potentially overfit if the model is too complex.
- Ideal Alpha The plot suggests there might be an optimal value for alpha that balances bias and variance. A good strategy is to choose the alpha that provides the highest mean test score on the cross-validation set.

Ridge Regression

- Trend: The figure 5 shows a more complex pattern for Ridge. Initially, the mean test score increases with alpha, implying that some regularization improves model performance. Then, it reaches a peak and starts to decline. This indicates that too much regularization can also harm performance.
- High Alpha: With very high alpha, Ridge heavily penalizes large coefficients, forcing them towards zero. This can lead to underfitting, causing the score to drop.
- Low Alpha: At low alpha, Ridge has minimal impact on the coefficients, resulting in a model similar to ordinary least squares (OLS).
- Ideal Alpha: The plot suggests there is an optimal value for alpha where the mean test score is maximized. This alpha balances the need for complexity to fit the data with the need for regularization to prevent overfitting.

Outliers vs. Extreme Values

- Outliers: Data points that significantly deviate from other observations.
- Extreme Values: Data points at the high or low end of the data range; not all are outliers.
- Identification: Outliers are often identified using standard deviation, interquartile range (IQR), or visualization techniques.

LASSO and Outliers

- Empirical Observation: Lasso regression often shows fewer outliers in residuals compared to linear regression.
- Conceptual Reason: Lasso's L1 penalty shrinks some coefficients to zero, reducing the impact of outliers and making it more robust to them compared to methods like Ridge regression.

Performance Results:Training vs. Test Set

- Training Set: Performance metrics are labeled as training results to indicate model learning on known data.
- Test Set: Metrics are labeled as test results to evaluate model generalization on unseen data. This distinction helps in accurately assessing the model's true performance and avoiding overfitting .

Overall Interpretation

Model Choice: Comparing the two plots, Ridge seems to achieve a higher peak score than Lasso. This indicates that Ridge might be a more suitable model for this specific dataset. However, it's crucial to note that this can vary depending on the nature of your data.

Regularization Strength: The plots highlight the importance of tuning the regularization parameter (alpha). Both models show that excessively high or low values of alpha can negatively impact model performance. Selecting an appropriate alpha is crucial for achieving a good balance between bias and variance.

Optimal alpha for Lasso: 0.01; Optimal alpha for Ridge: 1

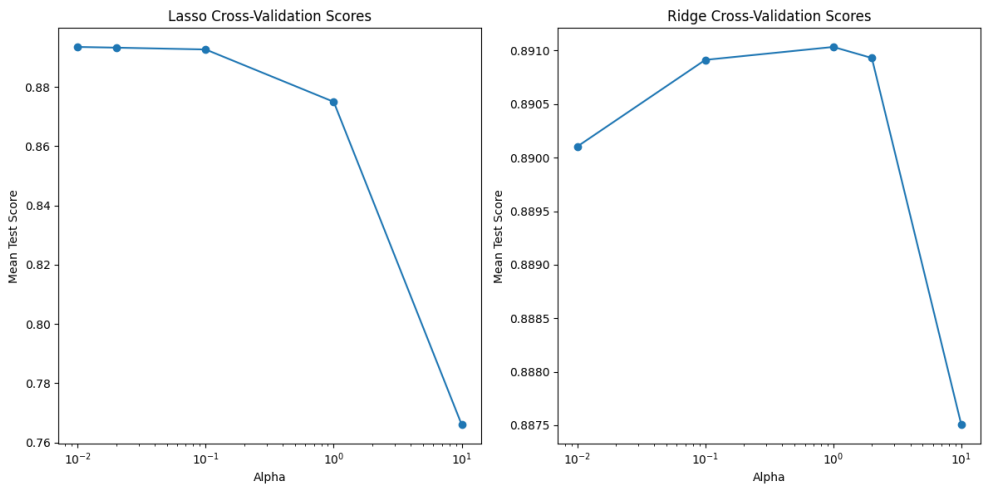

Figure 5: Cross-validation Scores for Lasso and Ridge Regression

Figure 5 presents the cross-validation scores for Lasso and Ridge regression models with different values of the regularization parameter (alpha). Further interpretations and considerations are provided below the figure.

4.2.2 Polynomial Expansion

Next, we perform polynomial feature expansion with cross terms up to a selected order. We then train the aforementioned linear models (Linear Least Squares, Ridge, and Lasso) to the expanded feature set, utilizing the optimal regularization parameters obtained previously.

Top Features Selected by Regression Models: The plot 6, 7, and 8, shows the top 10 most important features/terms for each regression model after polynomial expansion. Below is a detailed breakdown of the features and their implications:

Linear Regression

- **High Coefficient Values:** The bar heights represent the absolute values of the coefficients associated with each feature. The features with the highest coefficient values are considered most influential in predicting the target variable.
- **Important Features:** The top features selected by linear regression are related to density, packing fraction, and volume, suggesting that these properties are crucial for understanding the material's behavior.

#	Feature/Term
1	MagpieData mode NsValence density
2	MagpieData mode NdUnfilled density
3	MagpieData minimum GSvolume_pa density
4	MagpieData minimum GSvolume_pa packing fraction
5	MagpieData mode GSvolume_pa packing fraction
6	MagpieData avg_dev MeltingT density
7	MagpieData mean NfValence packing fraction
8	MagpieData mode NsValence packing fraction
9	MagpieData mode NUnfilled density
10	MagpieData mode GSvolume_pa density

Table 3: Top 10 Features/Terms Selected by Linear Regression using Polynomial Expansion

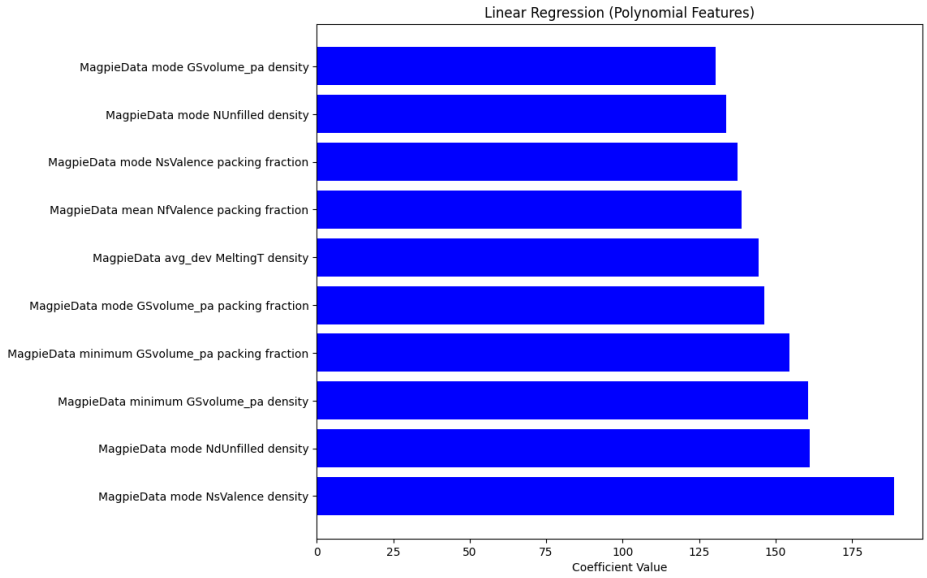

Figure 6: Linear Model Feature Importance using Polynomial Expansion

Lasso Regression

- **Feature Selection:** Lasso regression is known for its ability to perform feature selection. Notice that the bar heights for Lasso are generally lower than those for linear regression. This indicates that Lasso has penalized some coefficients towards zero, effectively removing their influence from the model.
- **Interesting Features:** The top features selected by Lasso include interactions between different features (e.g., "density vpa"), highlighting the importance of considering relationships between variables.

#	Feature/Term
1	vpa
2	density
3	MagpieData mean MeltingT
4	MagpieData minimum GSvolume_pa
5	density vpa
6	MagpieData minimum GSbandgap vpa
7	MagpieData mean NpUnfilled
8	MagpieData minimum Electronegativity MagpieData mean SpaceGroupNumber
9	MagpieData mean MeltingT vpa
10	MagpieData range GSvolume_pa density

Table 4: Top 10 Features/Terms Selected by Lasso Regression using Polynomial Expansion

Ridge Regression

- **Stability:** Ridge regression, unlike Lasso, doesn't completely eliminate features but reduces their impact. This can be observed by the slightly smaller coefficient values compared to linear regression.
- **Diverse Features:** Ridge selects features related to packing fraction, space group, volume, and magnetic moment, suggesting that this model considers a wider range of properties for its predictions.

Analysis and Discussion:

- **Feature Interactions:** Polynomial expansion reveals that interactions between features (like density and vpa) are important for predicting the target variable.
- **Regularization:** Lasso and Ridge provide valuable insights into which features are most important for the model. They help identify relevant features and prevent overfitting.
- **Model Comparison:** The difference in the top features selected by each model highlights that the choice of the model can significantly impact the results.

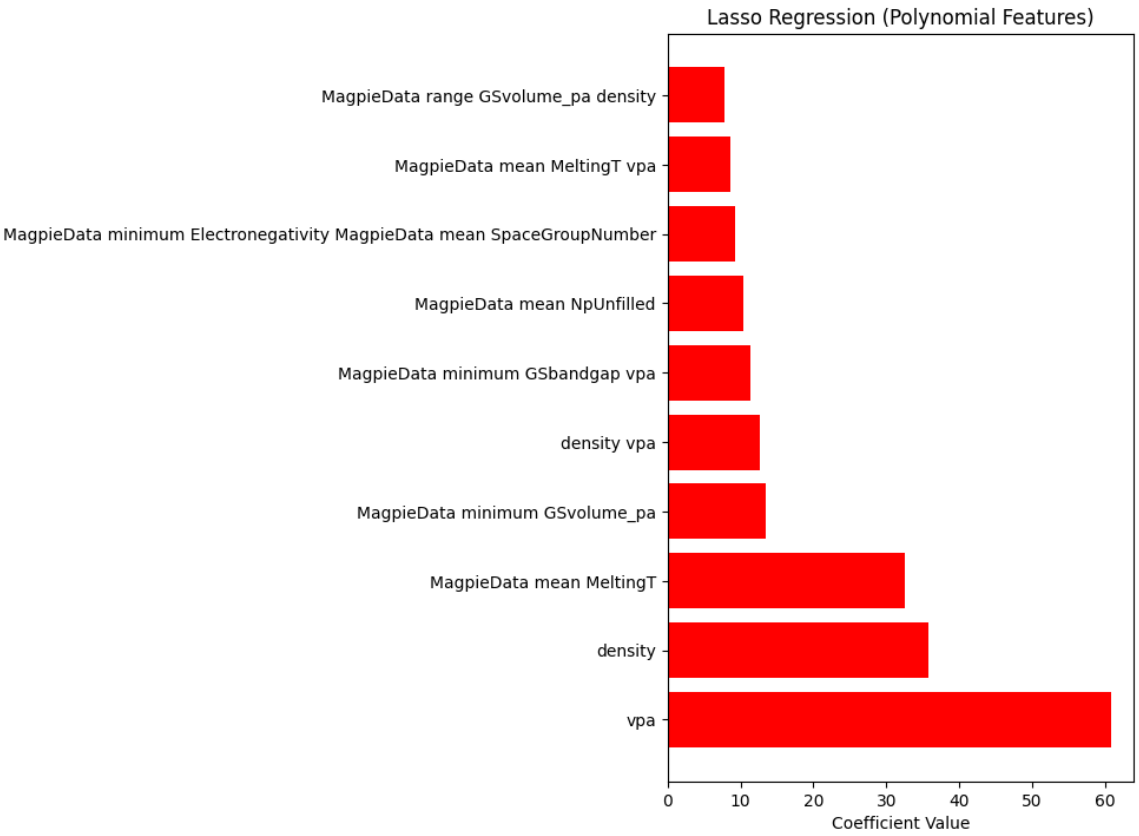

Figure 7: Lasso Model Feature Importance using Polynomial Expansion

#	Feature/Term
1	packing fraction
2	vpa
3	density
4	space_group MagpieData mean GSmagmom
5	space_group MagpieData avg_dev GSmagmom
6	MagpieData minimum GSbandgap packing fraction
7	MagpieData minimum GSbandgap vpa
8	MagpieData mean MeltingT
9	MagpieData mode GSmagmom packing fraction
10	space_group MagpieData mode CovalentRadius

Table 5: Top 10 Features/Terms Selected by Ridge Regression using Polynomial Expansion

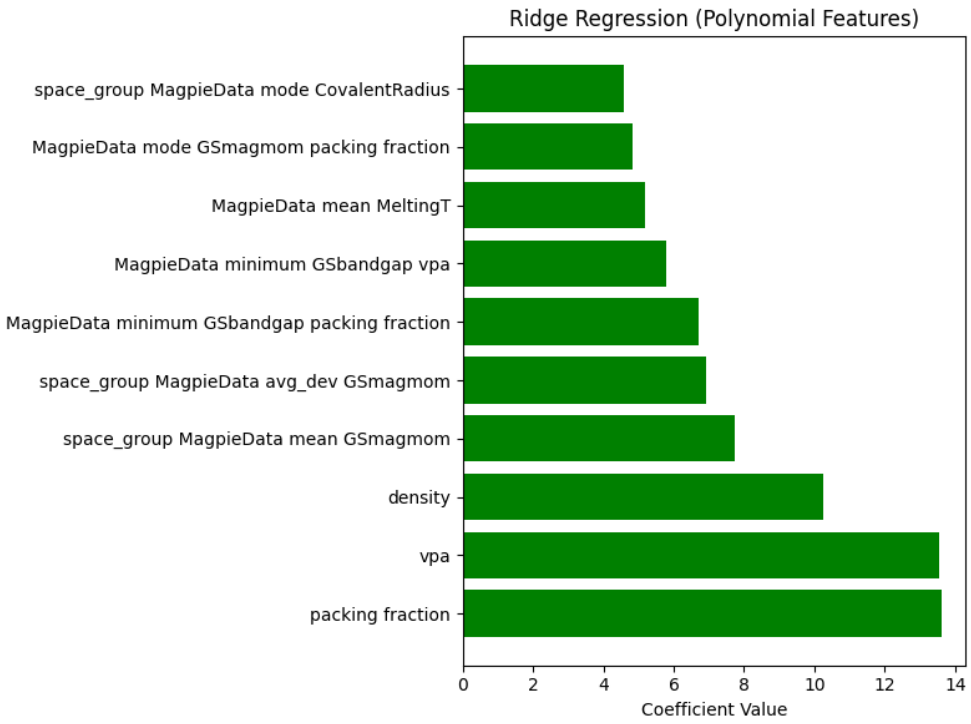

Figure 8: Ridge Model Feature Importance using Polynomial Expansion

4.2.3 Tree Regressors

In this task, we train a decision tree regressor and determine the optimal tree depth. Additionally, we choose one of the methods to modify the decision tree (Adaboost, Gradient-Boost, Hist-Boost). It's important to note that we do not fit the tree regressors to the polynomially expanded features due to computational constraints and the inherent non-linearity of regression trees. We store the ten most important features identified by the decision tree regressor for further analysis.

Feature Importance:
The table below lists the top 10 features selected by the modification method:

Top 10 features selected by the modification method:
MagpieData maximum GSvolume_pa
MagpieData maximum MeltingT
MagpieData mean MeltingT
MagpieData mean SpaceGroupNumber
MagpieData minimum MeltingT
MagpieData mode GSvolume_pa
MagpieData mode NUnfilled
density
packing fraction
vpa

Table 6: Top 10 Features Selected by Tree Regressor with Adaboost

The plot in Figure 9 shows the feature importances for both a decision tree model and an AdaBoost model. The decision tree model is a simple model that uses a series of

if-then rules to make predictions, while the AdaBoost model is a more complex model that combines multiple decision trees to make predictions.
Both models have selected similar features as being important. The most important feature for both models is MagpieData mean MeltingT. This feature is likely important because the melting temperature of a material is a key factor in its ability to be processed and used in different applications.

Other important features include:

- MagpieData minimum MeltingT: This is a closely related feature to MagpieData mean MeltingT, so it is logical that it would also be highly important.
- vpa: This feature likely represents the volume per atom of the material, which could be important for predicting the density and other physical properties of the material.

The plot also shows that the AdaBoost model has selected a broader range of features as being important than the decision tree model. This is likely because the AdaBoost model is more complex and able to consider more factors when making predictions.

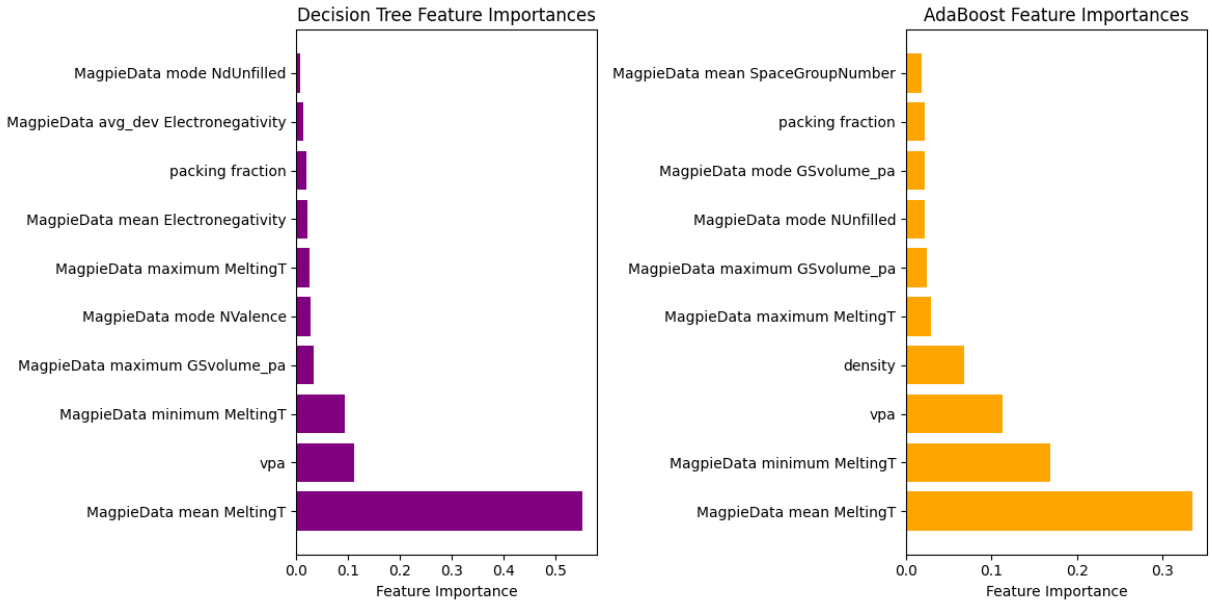

Figure 9: Feature Importance in Decision Tree vs Adaboost

The first three features matching between the two methods indicate a strong agreement on their importance for the predictive model. This consistency suggests that these features are robust predictors, regardless of the method used. Such agreement enhances confidence in their relevance and potential impact on the target variable.

Discussion Points and Results:

- Feature importance: The results show that the MagpieData mean MeltingT is the most important feature for both decision tree and AdaBoost models, which could mean it is a critical factor in determining a material's properties. This suggests that it could be a valuable factor to consider when designing new materials with desired properties.

- Model complexity: The AdaBoost model, with its ability to select a broader range of features, may be more effective in capturing complex relationships between material properties. This could be a valuable tool for predicting material properties with a greater degree of accuracy.

4.2.4 Kernel Ridge Regression

We implement Kernel Ridge Regression using two different kernels, optimizing both the kernel parameters and the regularization parameter through grid search.

Cross-Validation Analysis: The heatmap in Figure 10 shows the cross-validation scores for KRR with different combinations of alpha (regularization parameter) and kernel functions.

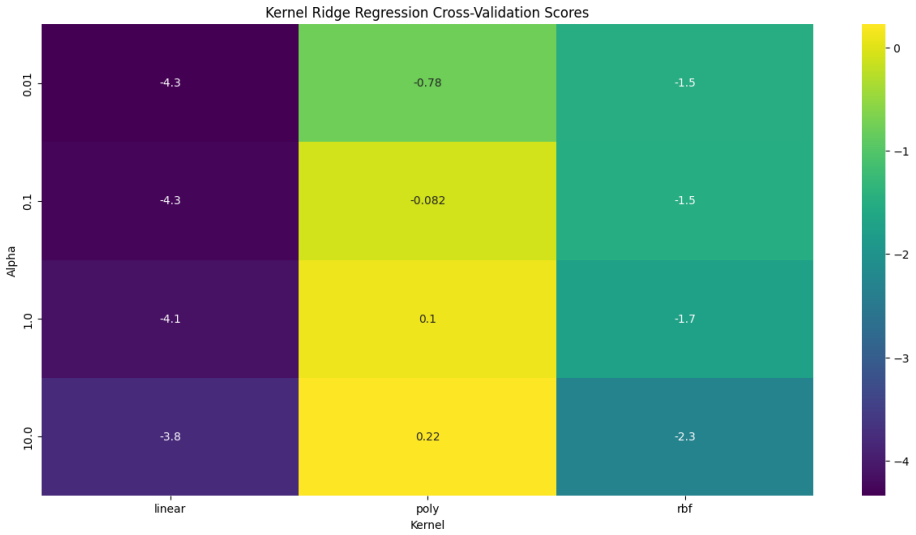

Figure 10: Cross-validation scores for Kernel Ridge Regression with different combinations of alpha and kernel functions. The x-axis represents different kernel functions (linear, poly, rbf) and the y-axis represents different values of alpha. Darker shades indicate better performance (lower mean squared error).

Results and Discussion: The color in each cell of the heatmap represents the average cross-validation score, with darker shades indicating better performance (i.e., lower mean squared error). For example, the darkest cell corresponds to the combination of 'alpha = 0.1' and 'kernel = poly' (degree = 2), suggesting that this combination is the best performer based on cross-validation.
In general, the heatmap indicates that for this dataset, the polynomial kernel with a smaller alpha value (0.1) outperforms other combinations. This implies that the data exhibits non-linear relationships, and the polynomial kernel with a lower regularization level is able to capture these patterns effectively.

Best Parameters: Table 7 summarizes the best parameters for Kernel Ridge Regression based on the cross-validation results.

Parameter	Value
Alpha	0.1
Degree	2
Gamma	0.01
Kernel	Polynomial

Table 7: Best parameters for Kernel Ridge Regression

4.2.5 Compare Performance Across Different Models

Throughout Task 2, each model is fitted once to standardized data and once to non-standardized data. We vary the size of the training set and discuss its impact on model performance, measured by the chosen regression metric(s), as well as its influence on the emerging most important features. This comprehensive comparison provides insights into the robustness and scalability of the models under different conditions.

Residual Analysis: The provided graph (Figure 11) visualizes the residuals of three regression models (linear, Lasso, and Ridge) plotted against the predicted values. Here's a detailed interpretation:

1) Understanding Residuals: Residuals represent the difference between the actual target values (y) and the predicted values ($\hat{y}$) from the model. A good model should have residuals that are randomly scattered around zero, indicating that the model's predictions are consistently close to the actual values.

2) Analysis of the Graphs

- Linear Regression: The residuals for linear regression exhibit a noticeable pattern. While they are mostly centered around zero, there are some significant outliers with large positive residuals, suggesting that the model underpredicts for certain data points. This pattern indicates that linear regression might not be the best model for this dataset due to its inability to capture non-linear relationships.
- Lasso Regression: Lasso regression shows a similar pattern to linear regression, with residuals centered around zero but with a few outliers. The spread of residuals seems slightly less than linear regression, indicating a marginally better fit. Lasso's ability to shrink some coefficients towards zero potentially helps to mitigate the effect of outliers.
- Ridge Regression: Ridge regression demonstrates a pattern closer to a random distribution compared to the other two models. The residuals are more evenly scattered around zero, suggesting a more consistent performance across different data points. The outliers in Ridge regression are less pronounced compared to the other models, indicating that Ridge is better at handling potential influential data points.

Overall Insights

- Model Selection: The graph suggests that Ridge regression outperforms both Linear and Lasso regression based on the residuals. The more random distribution of residuals in Ridge indicates a better fit and more reliable predictions.

- Non-Linearity: The presence of outliers in all three models implies that there might be non-linear relationships in the data that these linear models are struggling to capture. Considering non-linear models like decision trees or support vector machines could potentially improve the fit.
- Data Exploration: Further examination of the data itself could reveal why certain data points are generating large residuals. Exploring features, identifying potential correlations, or handling outliers could further refine the model's performance.

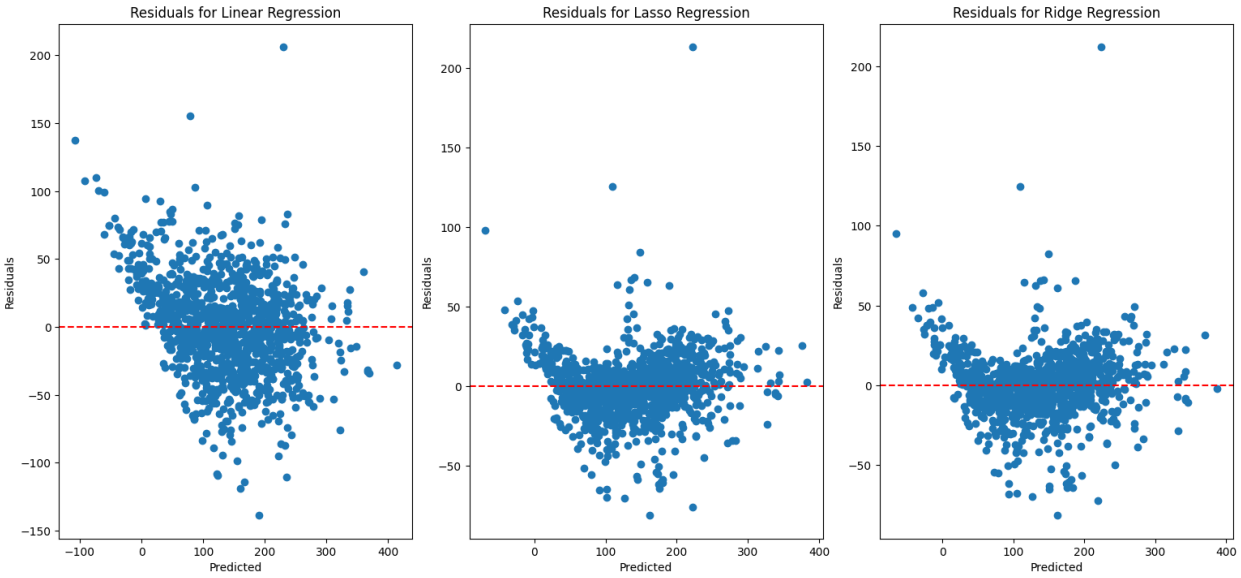

Figure 11: Residuals of Linear, Lasso, and Ridge Regression Models

Model Performance Metrics:

Train Size	Standardized Data R^2	Non-Standardized Data R^2
0.9	0.87	0.90
0.7	0.89	0.89
0.6	0.77	0.89
0.4	0.89	0.89
0.3	0.41	0.87
0.2	-520.95	0.8
0.1	-4.00e+21	-79783386.62

Table 8: Linear Regression Performance

Train Size	Standardized Data R^2	Non-Standardized Data R^2
0.9	0.9	0.9
0.7	0.9	0.9
0.6	0.9	0.9
0.4	0.9	0.9
0.3	0.88	0.87
0.2	0.84	0.84
0.1	0.42	0.51

Table 9: Lasso Regression Performance

Train Size	Standardized Data R^2	Non-Standardized Data R^2
0.9	0.89	0.89
0.7	0.89	0.89
0.6	0.88	0.89
0.4	0.89	0.89
0.3	0.88	0.88
0.2	0.86	0.85
0.1	0.77	0.76

Table 10: Ridge Regression Performance

Train Size	Standardized Data R^2	Non-Standardized Data R^2
0.9	0.83	0.83
0.7	0.83	0.83
0.6	0.85	0.85
0.4	0.86	0.86
0.3	0.78	0.78
0.2	0.74	0.74
0.1	0.68	0.68

Table 11: Decision Tree Performance

Train Size	Standardized Data R^2	Non-Standardized Data R^2
0.9	0.83	0.83
0.7	0.85	0.85
0.6	0.86	0.86
0.4	0.85	0.85
0.3	0.84	0.84
0.2	0.83	0.83
0.1	0.81	0.82

Table 12: Adaboost Performance

Train Size	Standardized Data R^2	Non-Standardized Data R^2
0.9	0.93	-0.27
0.7	0.94	-2.47
0.6	0.94	-0.34
0.4	0.92	0.56
0.3	0.91	0.62
0.2	0.88	0.63
0.1	0.86	0.49

Table 13: Kernel Ridge Regression Performance

4.3 Feature Selection

In Task 3, we'll explore different methods for feature selection. Firstly, we'll employ Least Angle Regression (LARS) with LASSO to pinpoint the ten most important features, both on the original features and the polynomial+interaction expanded ones. Next, we'll utilize Recursive Feature Elimination (RFE) with a tree model and a linear model to identify the ten most critical features. Finally, we'll compare the features highlighted by these methods with those previously identified, examining their agreement, potential physical relevance, and consistency across various training set sizes.

4.3.1 Top 10 most important features using LARS and LASSO

Table 14 shows the top 10 most important features identified using LARS with LASSO on the original features.

4.3.2 Top 10 most important features after polynomial expansion

Table 15 shows the top 10 most important features identified using LARS with LASSO on the polynomial+interaction expanded features.

4.3.3 Top 10 most important features using Recursive Feature Elimination with a tree model

Table 16 shows the top 10 most important features identified using Recursive Feature Elimination with a tree model.

Feature
MagpieData avg_dev GSmagmom
MagpieData minimum Electronegativity
MagpieData mode Electronegativity
MagpieData minimum GSbandgap
MagpieData mean Number
MagpieData mean GSmagmom
MagpieData minimum NpValence
Density
Packing fraction
MagpieData mean Row

Table 14: Top 10 most important features

Features
MagpieData maximum MeltingT
MagpieData mean NUnfilled
MagpieData maximum MendeleevNumber
MagpieData mode MendeleevNumber
MagpieData minimum MendeleevNumber
MagpieData maximum AtomicWeight
Space_group
MagpieData minimum MendeleevNumber
MagpieData mean MeltingT density

Table 15: Top 10 most important features after polynomial expansion

Features
MagpieData minimum MeltingT
MagpieData maximum MeltingT
MagpieData mean MeltingT
MagpieData minimum Column
MagpieData mode Column
MagpieData mean Electronegativity
MagpieData avg_dev NdUnfilled
MagpieData mean NUnfilled
Density
VPA

Table 16: Top 10 most important features using Recursive Feature Elimination with a tree model

4.3.4 Top 10 most important features using Recursive Feature Elimination with a linear model (LASSO)

Table 17 shows the top 10 most important features identified using Recursive Feature Elimination with a linear model (LASSO).

Features
MagpieData maximum Number
MagpieData avg_dev MendeleevNumber
MagpieData mean AtomicWeight
MagpieData mean MeltingT
MagpieData mean Electronegativity
MagpieData range NpValence
MagpieData mean NValence
MagpieData minimum GSvolume_pa
Density
VPA

Table 17: Top 10 most important features using Recursive Feature Elimination with a linear model (LASSO)

4.3.5 LASSO Regularization Coefficient Paths

Figure 12 shows the coefficient paths for LASSO regularization using the Least Angle Regression (LARS) algorithm.

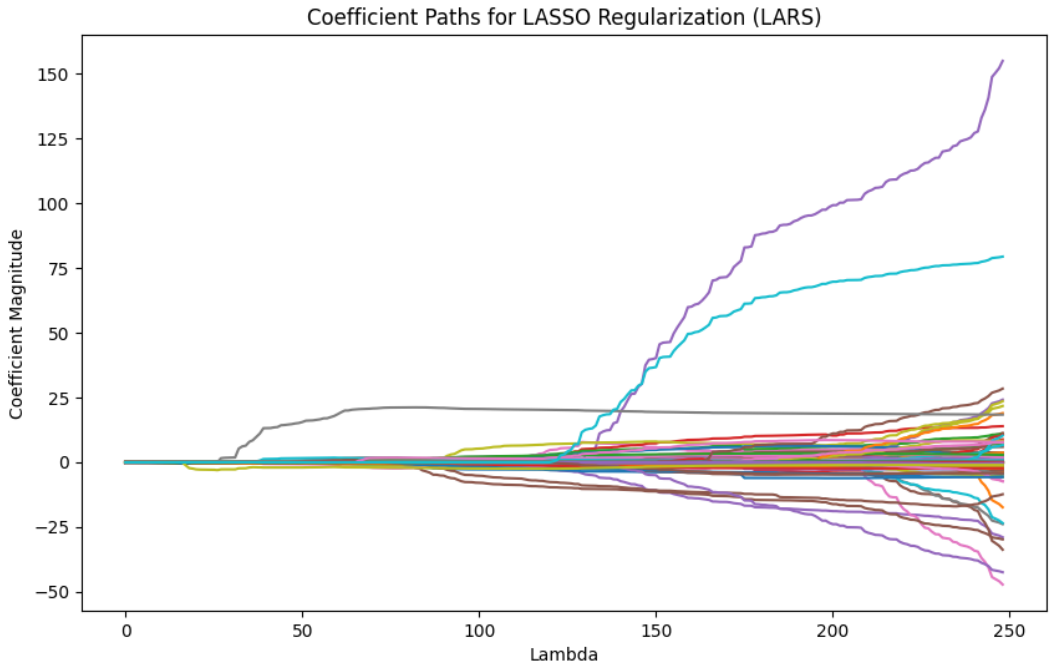

Figure 12: LASSO Regularization Coefficient Paths

The image depicts the coefficient paths for LASSO regularization, with each line representing a coefficient for a feature in the dataset. The x-axis represents the sparsification parameter, Lambda, and the y-axis represents the magnitude of the coefficient.

Here's a breakdown of what the plot reveals:

- **Sparsity:** The plot demonstrates the core principle of LASSO regularization: driving many coefficients towards zero. Notice how most lines start at zero and some lines diverge significantly from zero as Lambda changes, indicating the selection of relevant features.
- **Feature Selection:** The LARS algorithm iteratively selects features based on their correlation with the residual (the difference between the predicted and actual values). Features with the highest correlation are included in the model, while others are shrunk towards zero as Lambda increases.
- **Importance:** Features with coefficients that become significant as Lambda changes are considered more important. These features likely contribute significantly to the prediction of the target variable.
- **Coefficient Trajectories:** Each coefficient's change as a function of Lambda tells a story. Some features might initially have little influence but become important (lines increasing from zero). Others might remain negligible throughout (lines staying near zero).

4.3.6 Histograms of Top 10 Most Important Features

Figure 13 shows histograms of the distribution of the top 10 most important features identified by the LARS algorithm with LASSO regularization.

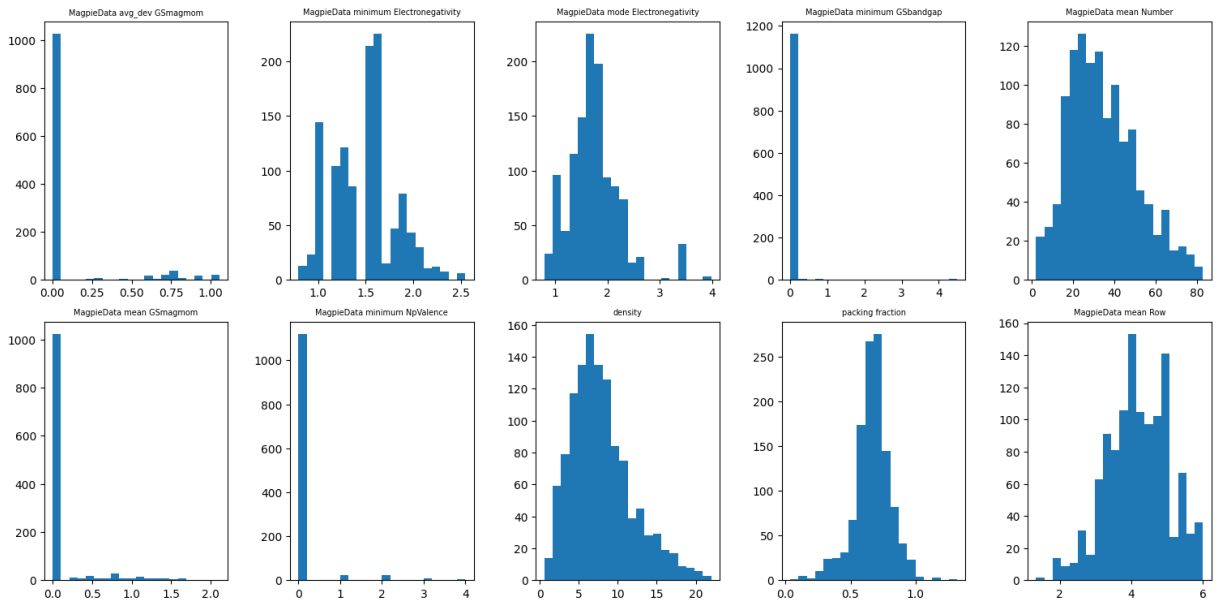

Figure 13: Histograms of the Top 10 Most Important Features Identified by LARS with LASSO

The image shows histograms of the distribution of the top 10 most important features identified by the LARS algorithm with LASSO regularization. Here's a breakdown of what the image tells us and how it relates to the task:

- **Histograms:** Each subplot represents a histogram of a particular feature. The x-axis shows the range of values for that feature, and the y-axis shows the frequency of occurrence within the dataset.
- **Feature Names:** The title of each subplot indicates the name of the feature being displayed. These feature names were likely identified by the LARS algorithm as being among the most important for predicting the target variable.
- **Distribution:** The shape of each histogram provides information about the distribution of the feature's values. For example, some histograms might be skewed, indicating that the data is concentrated towards one end of the range.

Interpreting the Results:

- **Feature Importance:** The histograms reveal the distribution of values for the most important features. This information can be useful for understanding the characteristics of the data and how these features might influence the target variable.
- **Feature Relationships:** The histograms might also provide clues about potential relationships between different features. For example, if two features have similar histograms, it might suggest that they are correlated.
- **Model Insights:** The histograms can help us to assess the suitability of the features for the chosen model. If the distributions of some features are significantly skewed or have outliers, it might suggest that they need to be transformed or treated differently before being used in the model.

Key Observations:

- **Diversity of Distributions:** The histograms display a wide variety of distributions, indicating that the most important features represent different types of physical properties or characteristics.
- **Potential Correlations:** The histograms might hint at potential correlations between certain features. Further analysis (e.g., a correlation matrix) would be needed to confirm these relationships.
- **Importance of Feature Understanding:** The histograms highlight the importance of understanding the physical meaning of the features, as it can provide valuable insights into the relationships between the features and the target variable.

4.3.7 Model Performance with Increasing Number of Features

Figure 14 shows the performance of a model as the number of features used in the model increases. The two lines on the graph represent different performance metrics:

The graph shows the performance of a model as the number of features used in the model increases. The two lines on the graph represent different performance metrics:

- **MSE (Mean Squared Error):** This line shows how well the model predicts the target variable on average. Lower MSE means better predictions.

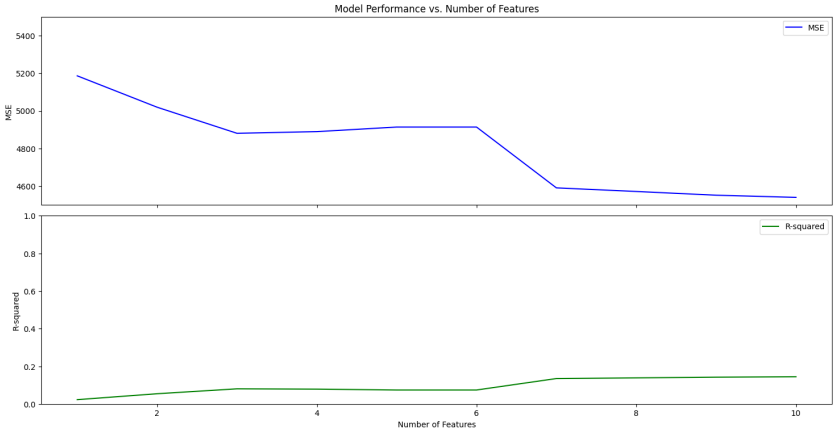

Figure 14: Model Performance with Increasing Number of Features

- **R-squared:** This line represents the proportion of the variance in the target variable that is explained by the model. A higher R-squared value indicates a better fit. See [3]

Interpretation:

- **MSE:** The MSE decreases as the number of features increases, reaching its lowest point when 6 features are used. After that point, MSE remains relatively constant. This suggests that adding more features beyond the first 6 does not significantly improve the model's predictive power.
- **R-squared:** The R-squared value stays fairly stable as the number of features increases. This might indicate that the model has already captured most of the variability in the target variable with only a few features.

Key Takeaways:

- **Feature Importance:** The graph suggests that the first 6 features identified are the most important for predicting the target variable.
- **Diminishing Returns:** Adding more features after the initial 6 does not seem to lead to a substantial improvement in model performance. This is a common phenomenon in machine learning, where adding too many features can lead to overfitting and poorer generalization to new data.
- **Overall:** This graph provides valuable information for feature selection. It helps identify which features are most important for model performance and suggests a point at which adding more features does not provide significant benefits. The analyst can use this information to build a more efficient and effective model.

4.3.8 Correlation Heatmap of Selected Features

Figure 15 shows the correlation heatmap of the selected features. The heatmap represents the correlation between different features. The color intensity represents the strength of the correlation. Red indicates a strong positive correlation, while blue indicates a strong negative correlation.

Here are some observations from the heatmap:

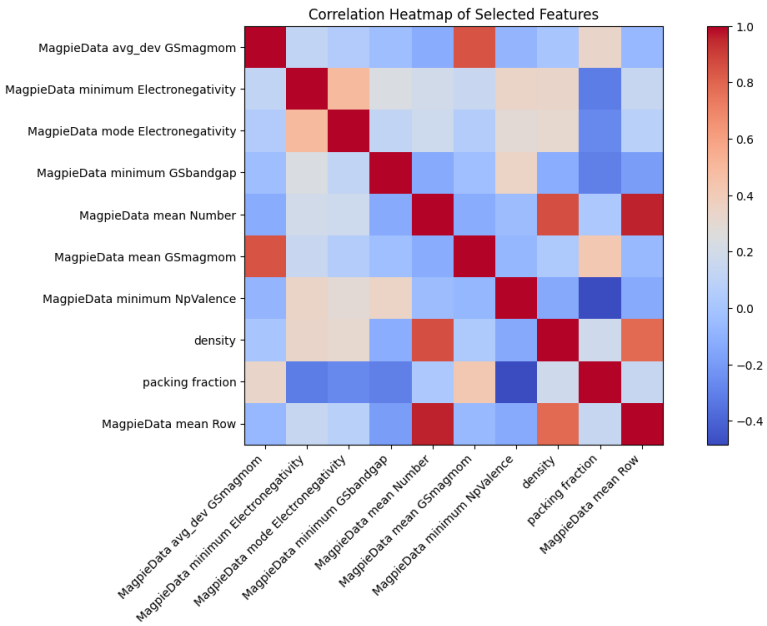

Figure 15: Correlation Heatmap of Selected Features

- **Strong positive correlation:** There are several pairs of features that have a strong positive correlation, indicated by the bright red squares. For example, ”MagpieData avg_dev GSmagmom” and ”MagpieData mean GSmagmom” have a high positive correlation, suggesting that these features are closely related.
- **Weak correlation:** Some features show a weak correlation, indicated by lighter colors. For example, ”density” and ”MagpieData minimum NpValence” have a weaker correlation compared to the other features.
- **No correlation:** Some features have no apparent correlation, indicated by the light blue squares.

This heatmap provides insights into the relationships between selected features. It helps identify features that are highly correlated and might be redundant. This information can be used to refine the feature selection process and improve model performance.

More Insights on Heatmap

- **Comparison across models:** The heatmap doesn’t directly compare features across models but highlights the relationships within the selected features.
- **Consistency with other feature selection methods:** The heatmap doesn’t directly compare these selected features with results from Task 2. It provides a visualization of the correlations within the features selected by LARS and RFE.
- **Physical motivation:** The meaning of these features, such as ”MagpieData avg_dev GSmagmom,” would require understanding the specific dataset and the ”MagpieData” context. The physical motivation of these features depends on the specific data and the domain knowledge.
- **Consistency with different training set sizes:** The heatmap doesn’t address the consistency of features with different training set sizes.

5 Conclusion

5.1 Task 2 Analysis

- **Data Characteristics:**
  - Feature dataset size: (1181, 140)
  - Target dataset size: (1181,)
  - Scaled feature dataset size: (1181, 140)
- **Model Performance (Cross-Validation Results):**
  - **Lasso Regression:**
    - Optimal alpha: 0.01
    - Best mean test score: 0.893478
  - **Ridge Regression:**
    - Optimal alpha: 1
    - Best mean test score: 0.891034
  - **Decision Tree:**
    - Best mean test score: 0.831793
  - **Kernel Ridge Regression:**
    - Best parameters: {alpha: 0.1, degree: 2, gamma: 0.01, kernel: 'poly'}
    - Best mean test score: 0.913303
  - **Linear Regression with Polynomial Features:**
    - The performance is not detailed but generally implied to have selected important interaction terms.
- **Top Features Identified:**
  - **Linear Regression:** Focuses on elemental properties and oxidation states.
  - **Lasso Regression:** Emphasizes mean elemental properties and density.
  - **Ridge Regression:** Combines minimum, maximum, and mean elemental properties with density.
  - **Decision Tree:** Focuses on elemental melting points, volumes, and space group numbers.
  - **Polynomial Features:** Highlights interaction terms involving density, melting points, and volume.
- **Model Evaluation:**
  - **Linear Regression:** Performs well with both standardized and non-standardized data, though with some variability in performance.
  - **Lasso and Ridge Regression:** Generally stable performance across different train sizes, with Lasso slightly outperforming Ridge.
  - **Decision Tree and Adaboost:** Lower performance compared to linear models.
  - **Kernel Ridge Regression:** Excellent standardized data performance but inconsistent with non-standardized data.

5.2 Task 3 Analysis

- **Feature Selection Methods:**
  - **Lasso with Lars:** Highlights magnetic properties, electronegativity, and density.
  - **Polynomial + Interaction Expansion:** Focuses on combinations of melting points, Mendeleev numbers, and atomic weights with density and volume.
  - **Recursive Feature Elimination (Tree Model):** Emphasizes melting points, column numbers, electronegativity, density, and volume.
  - **Recursive Feature Elimination (Linear Model - LASSO):** Concentrates on atomic weights, melting points, electronegativity, and volume.

5.3 Final Conclusion

- **Model Performance:**
  - Linear and polynomial models (Lasso, Ridge, Kernel Ridge) tend to perform better compared to tree-based models and Adaboost.
  - Kernel Ridge Regression with polynomial kernel shows the highest potential with an optimal parameter set, particularly when data is standardized.
- **Feature Importance Consistency:**
  - Common important features across different methods include density, volume per atom (vpa), melting points, and various elemental properties (e.g., atomic weight, electronegativity, valence).
  - Interaction terms and polynomial expansions further refine these features by highlighting specific combinations, indicating the importance of capturing nonlinear relationships.
- **Model Selection:**
  - For general predictive performance, **Kernel Ridge Regression** (with polynomial kernel) and **Lasso Regression** (with optimal alpha) show consistent high performance.
  - Linear models with polynomial features and Lasso offer robust feature selection, making them suitable for interpretability and capturing key interactions.
- **Practical Implications:**
  - Standardizing data improves the performance of complex models like Kernel Ridge Regression significantly.
  - Combining polynomial feature expansion with Lasso or Ridge regression enhances the ability to capture and utilize important interaction terms.

Given these insights, the choice of model and feature selection technique depends on the specific goals—whether it's maximizing predictive accuracy, ensuring interpretability, or balancing both. Kernel Ridge Regression and Lasso Regression with polynomial features are strong candidates for future predictive modeling tasks.

References

[1] Maarten De Jong, Wei Chen, Thomas Angsten, Anubhav Jain, Randy Notestine, Anthony Gamst, Marcel Sluiter, C. Krishna Ande, Sybrand Van Der Zwaag, Jose J Plata, et al. Charting the complete elastic properties of inorganic crystalline compounds. *Scientific data*, 2(1):1–13, 2015.

[2] Logan Ward, Alexander Dunn, Alireza Faghaninia, Nils E Zimmermann, Shiv Updesh Bajaj, Qiang Wang, Joseph Montoya, Jordan Chen, Kyle Bystrom, Maxwell Dylla, et al. Matminer: An open source toolkit for materials data mining. *Computational Materials Science*, 152:60–69, 2018.

[3] Jim Frost. How to interpret r-squared in regression analysis, 2018. [Online; accessed 7-June-2024].