Regression Analysis

Using Steel Industry Energy Consumption Dataset

Table of Contents

List of Figures

List of Tables

1 Background

The objective of this study is to select a public available dataset with at least 10 input and 1500 observations and conduct some statistical regression analysis in Python programming language. We selected the Steel Industry Energy Consumption Dataset Data Set (*UCI Machine Learning Repository: Steel Industry Energy Consumption Dataset Data Set*, 2021). This dataset contains electricity consumption information collected from the DAEWOO Steel Co. Ltd. in Gwangyang, South Korea. The steel facility produces different types of coils, steel plates and iron plates. Some relevant research publications that have utilized the dataset include (Sathishkumar *et al.*, 2020; VE, Shin and Cho, 2021).

The goals of this study include:

- Implement Linear Regression (LR) and regularized Support Vector Regression (SVR)
- Implement L1, L2 and Elastic Net (EN) regularization
- Implement Random Forest Regression (RFR) algorithm
- Provide a critical analysis of the models above, comparing and contrasting the models' comparative performance, interpretability and impact on linear regression coefficients.

2 Dataset

The dataset used contains energy consumption measurement for the steel company. There are 35040 observations and 11 variables including the date column. In this study, the task would be to predict the electricity consumption feature. A description of the dataset showing the variable name, column name abbreviation, variable type and measurement is provided in Table 1.

Table 1: A description of the dataset used in this study.

S/N	Input Variable	**Abbreviation**	**Type**	**Measurement**
1	Industry energy consumption (Target or dependent variable)	Usage	Continuous	kWh
2	Lagging current reactive power	Lag_RP	Continuous	kVarh
3	Leading current reactive power	Lead_RP	Continuous	kVarh
4	tCO2(CO2)	$CO2$	Continuous	Ppm
5	Lagging current power factor	Lag_PF	Continuous	%
6	Leading current power factor	Lead_PF	Continuous	%
7	Number of seconds from midnight	NSM	Continuous	Seconds
8	Week Status	WeekStatus	Categorical	Weekday/weekend
9	Day of the week (Mon, Tues, Wed, Thursday, Friday, Saturday, Sunday)	Day_of_week	Categorical	Sunday, Monday … Saturday
10	Load Type	Load_Type	Categorical	Light, medium or maximum load

As indicated in Table 1, the target variable is the electricity consumption, Usage. Figure 1 shows a plot of the actual steel industry energy consumption.

The experiments described in this study were carried out using the jupyter notebook. Thereafter, the notebook file was converted to a .py file. The file is best opened with a jupyter notebook.

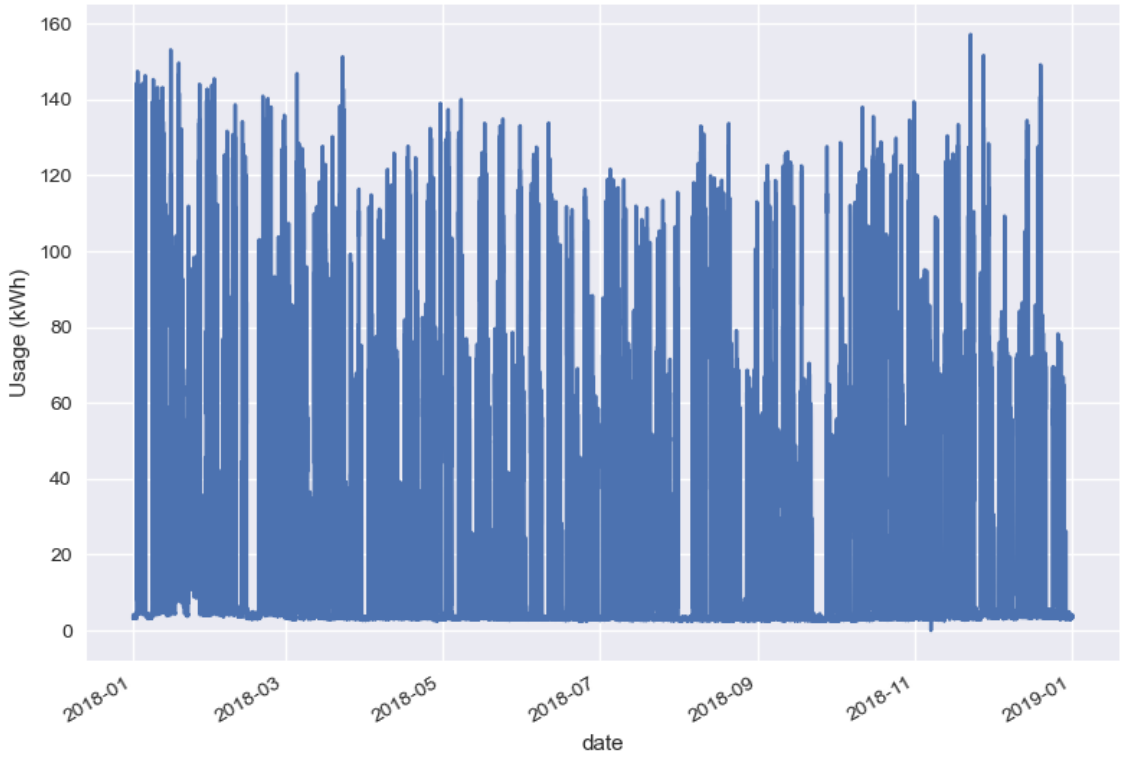

Figure 1: Steel industry energy consumption measurement for the whole study period.

3 Data Preparation

We begin by first preparing the dataset and getting it in the right format for Machine Learning (ML) purposes. Figure 2 below depicts the various stages involved in the data preparation process.

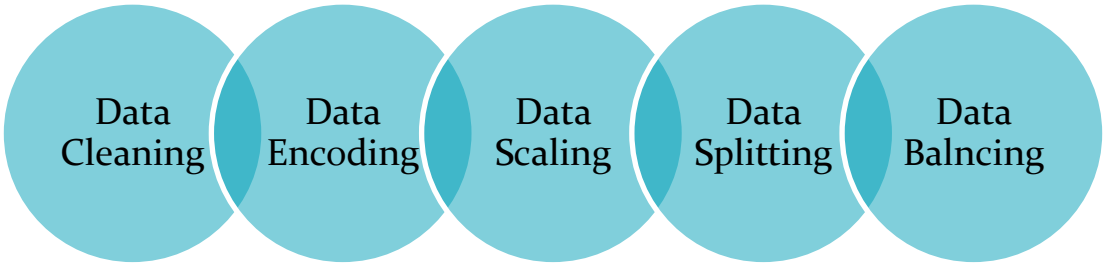

Figure 2: The Data Preparation Process

3.1 Data Cleaning

In the data cleaning stage of the dataset used in this work, we begin by checking if there are any null or missing values in our dataset. Figure 2 below shows that there are no missing values in any of the features in the dataset.

```
[32]: # Data preprocessing, checking for null and NaNs
      print(df.isnull().sum())

Usage          0
Lag_RP         0
Lead_RP        0
CO2            0
Lag_PF         0
Lead_PF        0
NSM            0
WeekStatus     0
Day_of_week    0
Load_Type      0
dtype: int64

[33]: print(df.isna().sum()) #check for NaN

Usage          0
Lag_RP         0
Lead_RP        0
CO2            0
Lag_PF         0
Lead_PF        0
NSM            0
WeekStatus     0
Day_of_week    0
Load_Type      0
dtype: int64
```

Figure 3: Checking for missing and NaN values

3.2 Data Encoding

Before we can apply the ML models to the dataset, we need to encode the three categorical variables shown in Table 1. This is because for ML applications, categorical data must be encoded into numeric values so that a number represents each categorical feature (Potdar, Pardawala and Pai, 2017).

```
1]: '''
    Convert categorical data to numerical

    One-hot encoding the categorical variables
    '''
    categorical_features = ['WeekStatus','Day_of_week','Load_Type']

    dataset = pd.get_dummies(df, columns=categorical_features)
    dataset.head()
```

Figure 4: Feature Encoding using One-Hot Encoding.

In Figure 4, we encode all three categorical variables using One-Hot Encoding technique (Yu *et al.*, 2022). We now have a dataset of 35040 observations and 19 input variables.

```
: print(dataset.shape)

  (35040, 19)
```

Figure 5 shows the correlation heat map of the dataset drawn using plotly.

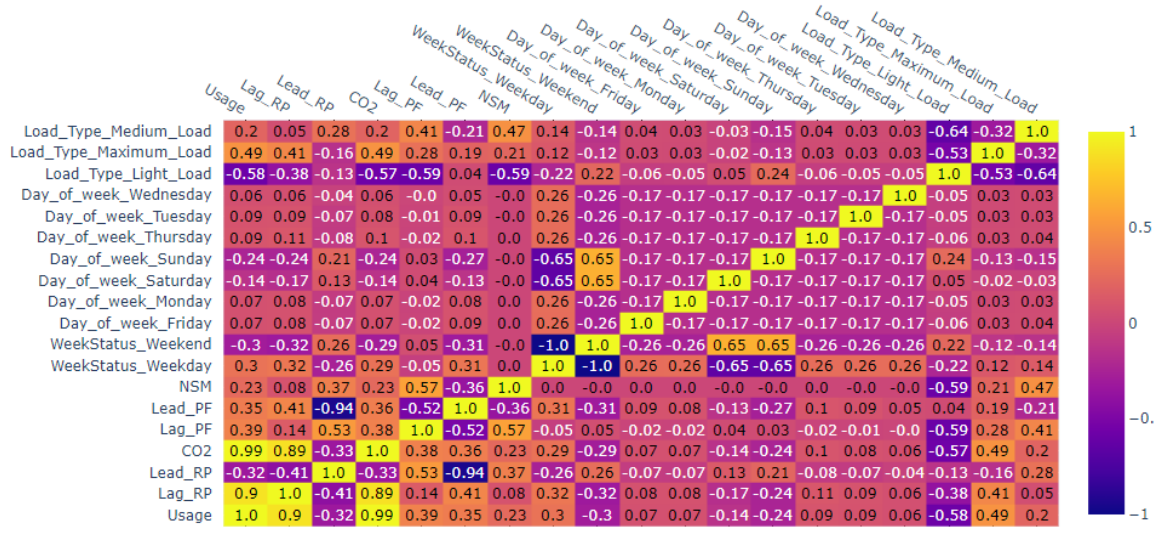

Figure 5: Heat map showing the feature correlation.

3.3 **Data Scaling, Splitting and Balancing**

In continuation of the data preparation process, we need to separate the target column from the feature set.

X = dataset.drop('Usage', axis = 1) # Features

y = dataset['Usage'] # Target Feature

Next we scale the dataset in order to prevent a feature from dominating other features. We use the Scikit-Learn StandardScaler function to scale the dataset so that each feature has a mean of 0 and a variance of 1.

X_scaled = StandardScaler().fit_transform(X) # scaling

We split the dataset into 70% to 30% for training and testing dataset respectively. The training datasets would be used for the model implementation. We would use the testing datasets for validating the performance of the models.

X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size = 0.3, random_state = 42)# splitting

3.4 Model Evaluation Metrics

To evaluate the performance of the regression models, we have written a helper function to compute the Mean Absolute Error (MAE), Mean Squared Error (MSE) and R-squared metrics. This function is shown in Figure 6 below.

```
# Using a helper function to print model metrics
def print_metrics(y_test, pred):
    mae = mean_absolute_error(y_test, pred)
    mse = mean_squared_error(y_test, pred)
    r2 = r2_score(y_test, pred)
    print('MAE:', mae)
    print('MSE:', mse)
    print('R2:', r2)
    print('')
```

Figure 6: Helper function for computing MAE, MSE and R-squared metrics.

In addition, we also calculate the Adjusted R-squared metrics using the function depicted in Figure 7.

```
# creating a function to create adjusted R-Squared

def adj_r2(X, y, model):
    r2 = model.score(X, y)
    n = X.shape[0]
    p = X.shape[1]
    adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

    return adjusted_r2
```

Figure 7: Function for calculating the Adjusted R-squared.

4 Training the ML Models

4.1 Least Squares Method

The standard linear regression model is of the form

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon \tag{1}$$

Where Y is the response variable and $X_1, X_2, \ldots X_p$ refer to a set of input variables. Typically, we can fit this sort of model using the least squares method (Dismuke and Lindrooth, 2006). Although a linear model fitted using this method has a unique advantage in terms of inference, there are some instances where alternative model fitting procedures may be preferred.

4.1.1 Prediction Accuracy

Linear models fitted using the least square estimates will have a low bias as long there is a linear relationship between the target variable and the input variables. If the number of observations is significantly more than the number of input variables, then the least squares method will have low variance and generalize well on unseen observations. However, if the number of observations are significantly more than the number of features or input variables, there will be a lot of variance in the least squares model and might cause the models to overfit. Models resulting from this fit will not perform well when presented with

testing or unseen data. To fix this problem, we need to constrain or shrink the estimated coefficients to reduce the model variance at the cost of an increase in the model bias (Friedman, 1997). This boosts the model accuracy and ensures that the fitted model generalizes well to new data.

4.1.2 Interpretability

There is a good chance that the input variables used in fitting a multiple regression model may be highly correlated with the target or response variable. This increases the model complexity when we include unnecessary features during the fitting process. If we remove these unwanted features by forcing the corresponding coefficients to zero, we stand a chance of generating models that are highly interpretable. However, the problem with the least squares method is that it is highly unlikely that there will ever be regression coefficients that are zero. As such, all the features end up been used in fitting the models. Again, this is where shrinkage methods or regularization is used.

4.1.3 Alternatives to the Least Squares Method

There are a number of alternatives used to overcome the limitations of the ordinary least squares method. These include subset selection (Miller, 2002), shrinkage or regularization (Emmert-Streib and Dehmer, 2019) and dimension reduction (Reddy *et al.*, 2020).

The two main shrinkage or regularization techniques are the least absolute shrinkage and selection operator (Lasso) and Ridge models. Both models are particularly useful for creating parsimonious models when dealing with a large number of input variables or features.

5 Impact of L1, L2, and Elastic Net (EN) Regularization on Linear Regression Coefficients

5.1 Linear Regression Model

To explore the impact of L1, L2 and EN regularization on the LR coefficients, we first fit a baseline LR model using the default settings with no regularization. Figure 8 depicts the model fitting and prediction processes using training and testing data.

```
# Fitting the LR model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# Prediction using the testing and training data
lr_pred = lr.predict(X_test)
lr_pred_train = lr.predict(X_train)
```

Figure 8: Fitting a baseline LR model.

Next, we extract the LR coefficients and sorting in ascending order in Figure 9.

	Features	Coefficients
15	Load_Type_Light_Load	1.827896e+13
17	Load_Type_Medium_Load	1.636315e+13
16	Load_Type_Maximum_Load	1.483311e+13
6	WeekStatus_Weekday	1.305982e+12
11	Day_of_week_Sunday	1.025553e+12
10	Day_of_week_Saturday	1.025553e+12
2	CO2	2.627583e+01
0	Lag_RP	5.567480e+00
3	Lag_PF	2.388292e+00
4	Lead_PF	1.889144e+00
1	Lead_RP	4.930954e-01
5	NSM	-8.230445e-02
13	Day_of_week_Tuesday	-5.756833e+11
14	Day_of_week_Wednesday	-5.756833e+11
8	Day_of_week_Friday	-5.756833e+11
12	Day_of_week_Thursday	-5.756833e+11
9	Day_of_week_Monday	-5.802632e+11
7	WeekStatus_Weekend	-7.618654e+11

Figure 9: Baseline LR model coefficients.

It is easy to notice that none of the model coefficients has been forced to zero; consequently, all input variables were used in fitting the LR model. It is also evident that the size of some coefficients are exponentially high and would tend to increase if the model complexities increases too. Even though that there are some variables that are not supposed to be used, the model fits all input variables.

5.2 L2 Regularization (Ridge Regression)

The linear model introduced in Equation 1 seeks to minimize the residual sum of squares (RSS) of the LR model. It does this with estimating the regression coefficients using the equation

$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2. \tag{2}$$

Ridge regression performs L2 regularization by adding a shrinkage penalty that has the effect of attempting to shrink the model coefficients towards zero. The ridge regression model estimates the model coefficients using the following minimization objective

$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2. \tag{3}$$

In Equation 3, the parameter λ is a tuning parameter we determine using cross-validation. The other term, $\lambda \sum_{j=1}^{p} \beta_j^2$, is the shrinkage penalty which shrinks the coefficients towards zero. When the tuning parameter, $\lambda = 0$, the penalty term has no effect and the Equation 3 defaults to the RSS computed from Equation 2. However, as the penalty grows towards infinity, i.e., $\lambda \longrightarrow \infty$, the shrinking effect grows and the regression coefficients will tend towards zero.

We fit a ridge regression model and pass in a grid of values for the tuning parameter as illustrated in Figure 10. We use the cross-validation to obtain the ideal value for our dataset.

```
%%time
r_lr = RidgeCV(alphas=[.001, .0015, .01, .015, .1, 1, 10, 100], cv=5)
r_lr.fit(X_train, y_train)
```

```
Wall time: 554 ms
```

```
RidgeCV(alphas=array([1.0e-03, 1.5e-03, 1.0e-02, 1.5e-02, 1.0e-01, 1.0e+00, 1.0e+01,
       1.0e+02]),
        cv=5)
```

Figure 10: Fitting a Ridge Regression model (L2)

Next, like before as we did for the LR model, we extract and sort the regression coefficients in ascending order in Figure 11. We observe a huge difference from the values shown for the LR regression coefficients in Figure 9. The L2 regularization has drastically reduced the magnitude of most of the input variables. We also observe that some have been shrunk towards zero. However, the model does not turn-off any feature, i.e., there is no regression coefficient that has been forced to zero. All features have been used in fitting the ridge regression model.

	Features	Coefficients
2	CO2	26.195775
0	Lag_RP	5.618200
3	Lag_PF	2.425294
4	Lead_PF	1.916488
1	Lead_RP	0.504587
17	Load_Type_Medium_Load	0.418656
13	Day_of_week_Tuesday	0.200145
16	Load_Type_Maximum_Load	0.149704
11	Day_of_week_Sunday	0.059179
7	WeekStatus_Weekend	0.009838
6	WeekStatus_Weekday	-0.009838
14	Day_of_week_Wednesday	-0.025689
10	Day_of_week_Saturday	-0.046475
9	Day_of_week_Monday	-0.053788
8	Day_of_week_Friday	-0.056036
12	Day_of_week_Thursday	-0.076908
5	NSM	-0.078249
15	Load_Type_Light_Load	-0.496259

Figure 11: Ridge Regression (L2) coefficients

5.3 L1 Regularization (Lasso)

Our analysis in the previous section on L2 regularization revealed There is an obvious problem of having to include all the input features when implementing L2 regularization. The shrinkage penalty of the ridge regression model introduced in Equation 3 will shrink the coefficients towards zero (depicted in Figure 11) but will not succeed in setting any to zero unless $\lambda = \infty$. This presents a problem to the model interpretability especially if the features are large. All features in our model which do not contribute to the prediction accuracy have all been used in the model fitting process.

The Lasso performs L1 regularization in which the shrinkage penalty has the desired effect of succeeding in removing some input features from the model. The Lasso model is of the form

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|. \tag{4}$$

L1 regularization via the Lasso forces some coefficients to zero and introduces sparsity in the model when the shrinkage penalty, λ is sufficiently large.

We demonstrate this process by fitting a Lasso model. Figure 12 illustrates the regression coefficients obtained from L1 regularization. We can see that some input variables have zero regression weights. These is an effect of the L2 regularization, which removes unwanted variables by shrinking the coefficients towards zero or setting them to zero. In the end, Lasso succeeds in producing sparse models. This improves the interpretability of the models.

	Features	Coefficients
2	CO2	26.286720
0	Lag_RP	5.556616
3	Lag_PF	2.392119
4	Lead_PF	1.883480
1	Lead_RP	0.497963
13	Day_of_week_Tuesday	0.232266
17	Load_Type_Medium_Load	0.156098
11	Day_of_week_Sunday	0.103558
14	Day_of_week_Wednesday	0.005542
7	WeekStatus_Weekend	0.000000
6	WeekStatus_Weekday	-0.000000
10	Day_of_week_Saturday	-0.000000
9	Day_of_week_Monday	-0.020366
8	Day_of_week_Friday	-0.022734
12	Day_of_week_Thursday	-0.043580
5	NSM	-0.077550
16	Load_Type_Maximum_Load	-0.086301
15	Load_Type_Light_Load	-0.778237

Figure 12: Lasso Regression coefficients. Some features coefficients are zero.

5.4 Elastic Net

The Lasso algorithm has poor prediction accuracy due to high correlations between features, especially when the number of observations is greater than the feature space. The EN combines both the Lasso and ridge regression, that is, it applies a mixture of $\ell 1$-norm and $\ell 2$-norm penalties on the coefficients. The EN algorithm is a penalized LR model that incorporates the $\ell 1$ as well as $\ell 2$ penalties during training.

Let us review the effect of the EN model on the coefficients for our dataset in Figure 13. Similar to the Lasso, we observe from Figure 13 that the EN model succeeds in setting at least one coefficient to zero. In other cases, it does shrink the other coefficients towards zero as well. This helps improve model interpretability over models with no regularization.

	Features	Coefficients
2	CO2	26.135132
0	Lag_RP	5.656583
3	Lag_PF	2.443392
4	Lead_PF	1.916427
1	Lead_RP	0.488498
13	Day_of_week_Tuesday	0.225100
17	Load_Type_Medium_Load	0.217589
11	Day_of_week_Sunday	0.082139
7	WeekStatus_Weekend	0.007362
14	Day_of_week_Wednesday	0.000000
6	WeekStatus_Weekday	-0.015646
10	Day_of_week_Saturday	-0.023079
9	Day_of_week_Monday	-0.027820
8	Day_of_week_Friday	-0.030286
16	Load_Type_Maximum_Load	-0.031907
12	Day_of_week_Thursday	-0.050930
5	NSM	-0.077694
15	Load_Type_Light_Load	-0.731352

Figure 13: The EN regression coefficients.

In order to compensate for the over-fitting and overly optimistically high magnitude coefficients that ordinary regression can provide, elastic net deliberately penalize regression coefficients. The result of this is that the EN effectively shrinks some coefficients, as is the case with ridge regression and sets some to zero (as in Lasso).

6 Impact of L2 Regularization on Support Vector Regression (SVR) Performance and Interpretability

6.1 Baseline SVR Model

In Figure 14, we begin by first fitting a baseline SVR model with no regularization parameters used.

```
svr = SVR()
```

```
grid_param = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}

gd_svr = GridSearchCV(estimator=svr, param_grid=grid_param, scoring='r2', cv=5)

gd_svr.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=SVR(),
             param_grid={'kernel': ['linear', 'poly', 'rbf', 'sigmoid']},
             scoring='r2')
```

```
svr_gs = gd_svr.best_estimator_  ## get the best estimator model
```

Figure 14: Fitting a Baseline SVR model.

In Figure 15, using the unseen testing data and training data, we report the baseline SVR model performance with respect to the MAE, MSE and R-squared metrics.

```
print('Metrics for testing data:\n')
print_metrics(y_test, svr_pred)
print('metrics for training data:\n')
print_metrics(y_train, svr_pred_train)
```

```
Metrics for testing data:

MAE: 1.847312611957154
MSE: 12.784172497781764
R2: 0.988609797517273

metrics for training data:

MAE: 1.8851770798799274
MSE: 15.601631287562626
R2: 0.9860298945876468
```

Figure 15: Baseline SVR results.

6.2 Regularized SVR Model

Next, we fit a regularized SVR model to evaluate its performance compared with the baseline SVR model. Machine learning algorithms that use the kernel method have a tendency to overfit because they are translating data into higher-dimensional regions (Tian and Zhang, 2022). Regularization parameters are useful for managing this as usual. Gamma and C functions are made available for this use by the Scikit-learn SVM techniques. The gamma value serves as a multiplier within the kernel computation. The larger the gamma value, the more closely the SVR model will attempt to approximate the data.

In Figure 16, we show the SVR model fitting with regularization applied.

Tuning the SVR parameters 'kernel', 'C', 'epsilon' and implementing cross-validation using Grid Search

```
svr_reg = SVR()
grid_param = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid'], 'C': [100,1000,10000], 'epsilon': [100,1000,10000],
              'gamma': ['auto',1,10,100]}

gd_svr_reg = GridSearchCV(estimator=svr_reg, param_grid=grid_param, scoring='r2', cv=5)

gd_svr_reg.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=SVR(),
             param_grid={'C': [100, 1000, 10000], 'epsilon': [100, 1000, 10000],
                         'gamma': ['auto', 1, 10, 100],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']},
             scoring='r2')
```

```
print("SVR with regularization best parameters: ", gd_svr_reg.best_params_)
```

```
SVR with regularization best parameters:  {'C': 100, 'epsilon': 100, 'gamma': 'auto', 'kernel': 'linear'}
```

Figure 16: Fitting a Regularized SVR model.

Interestingly, we observe that applying regularizing to the SVR model does not improve the model performance or interpretability. The SVR model with regularization performs poorly compared with the baseline SVR model. The result is shown in Figure 17.

```
: print('Metrics for testing data:\n')
  print_metrics(y_test, svr_reg_pred)
  print('metrics for training data:\n')
  print_metrics(y_train, svr_reg_pred_train)
```

```
Metrics for testing data:

MAE: 55.375447108066965
MSE: 3717.6717898972606
R2: -2.3123015555836237

metrics for training data:

MAE: 55.78271811806915
MSE: 3749.930432391553
R2: -2.357784994653945
```

Figure 17: SVR model with regularization.

Generally, Support-vector machines (SVMs) can be understood in the context of other regularization-based machine-learning methods by using regularization perspectives on SVMs. Binary data is categorized using SVM algorithms with the aim of minimizing the average of the hinge-loss function and the L2 norm of the learnt weights. This approach minimizes the bias and variance of our estimator of the weights while also avoiding overfitting via Tikhonov regularization (Djennadi, Shawagfeh and Arqub, 2021) and in the context of the L2 norm.

7 Random Forest Regression (RFR) versus all models including SGD Regressor

We use a non-linear method, the RF algorithm, which is an ensemble method. The RF algorithm constructs multiple DTs and combine their output in order to make stable and more accurate predictions.

We compare the performance of the RFR model with the regularized techniques: EN, Lasso, RR and the baseline LR model, baseline SGDRegressor (no penalty) and SGDRegressor with the EN penalty. Figure 18 shows the regression coefficients for all models.

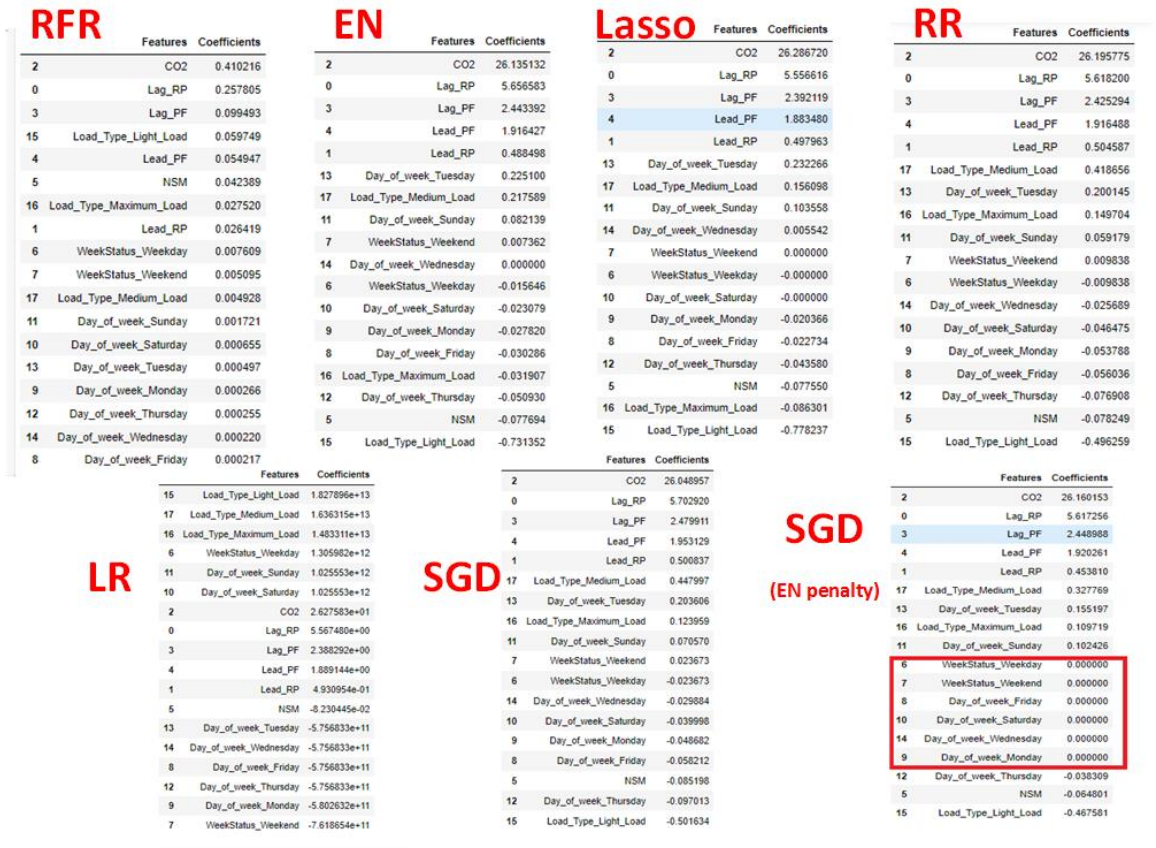

Figure 18: Comparing the RFR model with baseline LR models and the regularized linear models.

We observe that the SGDRegressor model with the EN penalty achieves the best interpretable model having being able to set about six weights to zero as shown in the illustration. During the model-training phases of both models, we applied a range of techniques to prevent the models from overfitting.

On the other hand, the RFR algorithm achieves the best shrinkage as it succeeds in shrinking almost all coefficients very near to zero. This aids interpretation too. Moreover, when we compare the model performance for all techniques examined, the RFR algorithm achieves the best performance. We show the results of all the models with respect to the MAE, MSE and R-squared in Figure 19.

	MAE	MSE	R2
Linear Regression	2.553545	17.774194	0.984164
SGD Regression	2.568298	17.867094	0.984081
SGD Regression with EN penalty	2.541696	17.791191	0.984149
SVR Baseline	1.847313	12.784172	0.988610
SVR with Regularization	55.375447	3717.671790	-2.312302
Ridge Regression	2.558148	17.804217	0.984137
Lasso Regression	2.551680	17.766536	0.984171
Elastic Net	2.562649	17.829159	0.984115
Random Forest Regression	1.015281	3.994391	0.996441

Figure 19: The performance of all the models in MAE, MSE and R-squared.

We can see that the RFR model (highlighted with the green box) produces the best performance with respect to all three-evaluation metrics. The SVR model recorded the worst performance (highlighted in red).

It is interesting to note that while the SGDRegressor model with EN penalty achieves the best sparse model which aids interpretability of the prediction results, the RFR model records a better model performance in all three regression evaluation metrics evaluated.

8 Conclusion

The RFR models produced better prediction performance when compared with the regularized linear models. However, the regularized linear models especially the SGDRegressor with EN penalty and the Lasso produced sparse models, which are more interpretable. The regularized models succeed in setting some input parameters to zero, thereby, reducing the number of features used in fitting the models and producing parsimonious models.

We can attribute the prediction performance of the random forest algorithm to its inherent structure. However, while growing trees, the random forest adds more randomness to the model. As such, instead of looking for the most significant feature when dividing a node, it looks for the best feature among a random subset of features. As a result, it minimizes the variation and solves the overfitting issue in decision trees, thereby increasing accuracy.

References

Dismuke, C. and Lindrooth, R. (2006) 'Ordinary least squares', *Methods and Designs for Outcomes Research*, 93, pp. 93–104.

Djennadi, S., Shawagfeh, N. and Arqub, O.A. (2021) 'A fractional Tikhonov regularization method for an inverse backward and source problems in the time-space fractional diffusion equations', *Chaos, Solitons & Fractals*, 150, p. 111127.

Emmert-Streib, F. and Dehmer, M. (2019) 'High-dimensional LASSO-based computational regression models: Regularization, shrinkage, and selection', *Machine Learning and Knowledge Extraction*, 1(1), pp. 359–383.

Friedman, J.H. (1997) 'On bias, variance, 0/1—loss, and the curse-of-dimensionality', *Data mining and knowledge discovery*, 1(1), pp. 55–77.

Miller, A. (2002) *Subset selection in regression*. chapman and hall/CRC.

Potdar, K., Pardawala, T.S. and Pai, C.D. (2017) 'A comparative study of categorical variable encoding techniques for neural network classifiers', *International journal of computer applications*, 175(4), pp. 7–9.

Reddy, G.T. *et al.* (2020) 'Analysis of dimensionality reduction techniques on big data', *IEEE Access*, 8, pp. 54776–54788.

Sathishkumar, V.E. *et al.* (2020) 'An energy consumption prediction model for smart factory using data mining algorithms', *KIPS Transactions on Software and Data Engineering*, 9(5), pp. 153–160.

Tian, Y. and Zhang, Y. (2022) 'A comprehensive survey on regularization strategies in machine learning', *Information Fusion*, 80, pp. 146–166.

UCI Machine Learning Repository: Steel Industry Energy Consumption Dataset Data Set (no date). Available at: https://archive.ics.uci.edu/ml/datasets/Steel+Industry+Energy+Consumption+Dataset (Accessed: 16 December 2022).

VE, S., Shin, C. and Cho, Y. (2021) 'Efficient energy consumption prediction model for a data analytic-enabled industry building in a smart city', *Building Research & Information*, 49(1), pp. 127–143.

Yu, L. *et al.* (2022) 'Missing data preprocessing in credit classification: One-hot encoding or imputation?', *Emerging Markets Finance and Trade*, 58(2), pp. 472–482.