Comparative Analysis of Regression Models

Written by: Jala Daniel, Eugene Lowe, Anirudh Vannemreddy, Jahnavi Chintala, Victor Mgbeafulu

Date: December 13, 2024

Course: Data Science and Analytics 6000

Abstract

This project aims to evaluate the performance of six regression models across 20 diverse datasets, focusing on model selection, hyperparameter tuning, and interpretation of model metrics. The primary objective was to identify the best regression techniques for various data complexities, while addressing challenges like overfitting, underfitting, and computational resource management. The study involved applying Linear Regression, Ridge Regression, Lasso Regression, Principal Component Regression (PCR), Support Vector Regression (SVR), and Regression Splines to each dataset, with and without hyperparameter tuning via cross-validation. Metrics such as MSE, RMSE, RSS, R^2, and RSE were used to assess model performance. Results showed that hyperparameter tuning significantly improved model accuracy, with Lasso Regression and Ridge Regression demonstrating notable reductions in error and improved model fit. The findings underscore the importance of model tuning and feature selection in enhancing predictive performance, particularly for complex datasets. This study contributes valuable insights into best practices for regression modeling in data science.

Background

The goal for this project was to gain a better understanding of how to build regression models, model selection, and how to interpret model metrics. The methodology for this project consisted of selecting 20 diverse datasets (e.g. different number of observations, predictors, complexity), data cleaning/preprocessing, exploratory data analysis, model building, metric evaluation. The largest challenges were managing our computational resources, time management, and dealing with overfitting or underfitting. While the challenges were difficult, this project will enforce best benchmarking practices while enhancing problem solving skills and increase the skillset for each team member's professional resume.

Methodology

Regression models were chosen to help explain the datasets. Different metrics were also used to measure the success of each model. Linear regression, Ridge regression, Lasso Regression, Principal Component Regression (PCR), Regression Splines and Support Vector Regression (SVR) were the choice of regression models for this study. Along with each model, hypertuning with cross-validation was performed on each model to optimize the models' hyperparameters, resulting in improvements in model performance. Here is a general rundown of the project approach:

- 20 non-related datasets were chosen to evaluate.
- Then, the 6 regression methods were performed without hypertuning / CV to generate the 5 metric results (MSE, RMSE, RSS, R^2, & RSE).
- Next, the 6 regression methods were performed again, this time with hypertuning / CV to generate the 5 metric results (MSE, RMSE, RSS, R^2, & RSE).
- Each method's results were then compared to it's respective hypertuning / cv results to determine whether hypertuning / cv improved model performance.
- Lastly, learning curve graphs were plotted.

What is regression Analysis?

In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the outcome or response variable, or a label in machine learning parlance) and

one or more error-free independent variables (often called regressors, predictors, covariates, explanatory variables or features).

Regression analysis is primarily used for two conceptually distinct purposes. First, regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Six different models were used to predict the target variable across 20 different datasets to determine which model performs best for each dataset.

To evaluate the performance of the models in predicting the target variable, several evaluation metrics can be used. These metrics allow us to measure how well the models fit the data and generalize to unseen observations. The following section defines the metrics used and introduces the six models applied across the 20 datasets.

Evaluation Metrics and Formulas:

To assess the performance of the six regression models across the datasets, the following metrics were used:

1. Mean Squared Error (MSE)

Measures the average squared difference between the actual and predicted values. Lower MSE values indicate better model performance.

Formula: MSE = $\frac{1}{n}\Sigma(y_i - \hat{y}_i)^2$

Where:

- y_i: Actual value of the target variable$(א)^2$
- $\hat{y}_i$: Predicted value of the target variable
- n: Number of observations

2. Root Mean Squared Error (RMSE)

Provides error in the same units as the target variable, making it easier to interpret.

Formula: RMSE = $\sqrt{MSE}$

3. Residual Sum of Squares (RSS)

Represents the total deviation of the predicted values from the actual values. A lower RSS indicates a better fit.

Formula: RSS = $\Sigma(y_i - \hat{y}_i)^2$

4. Coefficient of Determination (R^2)

Indicates the proportion of the variance in the dependent variable that is explained by the independent variables. Values close to 1 indicate a good fit.

Formula: $R^2 = 1 - \frac{RSS}{TSS}$

Where:

- TSS: Total sum of squares, calculated as TSS = $\Sigma(y_i - \bar{y}_i)^2$
- $\bar{y}$: Mean of the actual target values

5. Residual standard Error:

Measures the standard deviation of the residuals. It provides an estimate of the average prediction error in the same units as the dependent variable. RSE helps to assess how well the model fits the data.

Formula: RSE = $\sqrt{\frac{RSS}{(n-p-1)}}$

Where:

- RSS: Residual Sum of Squares
- n: Number of observations
- p: Number of predictors

Models Used:

LASSO Regression:

LASSO or Least Absolute Shrinkage and Selection Operator, also called L1 regularization, is a regularization technique that adds a penalty to avoid overfitting, making the model more accurate in predictions. It minimizes the residual sum of squares with a penalty proportional to the absolute value of the magnitude of coefficients. This penalty can set some coefficients to zero, which is equivalent to doing feature selection.

Pros of LASSO Regression:

- It selects important features automatically and shrinks the irrelevant ones to zero.
- Reduces overfitting, which occurs from excessive complexity of the model.

Disadvantages of LASSO Regression:

- It may perform poorly in cases with highly correlated predictors because it only "picks" one variable and leaves the rest.

Ridge Regression:

Ridge Regression is a form of linear regression that employs L2 regularization, which adds a penalty term based on the magnitude of the coefficients to improve the overall stability of the model and help avoid overfitting. It conservatively shrinks the residual sum of squares (RSS) with a penalty per coefficient as the square of its magnitude. This penalty reduces coefficients towards 0 but does not get rid of them completely.

Benefits of Ridge Regression

- Works Well with Multicollinearity: Ridge Regression is quite useful when the predictors are highly correlated. It stabilizes the coefficient estimates and thus reduces their variance.
- Better Generalization: Ridge Regression shrinks the coefficients, thus minimizing the possibility that the model overfits and helps it generalize better on unseen data.
- Uses All Features: Ridge Regression keeps all coefficients, allowing all variables to remain in the model unlike LASSO.

Disadvantages of Ridge Regression

- Feature Selection: Ridge Regression does not do feature selection since it shrinks coefficients but does not remove them (unlike LASSO).
- Parameter Tuning: Needs proper tuning of the λ parameter and often requires cross-validation.
- Bias-Variance Trade-off: Excessive addition of regularization λ can potentially induce bias, hindering the model's capacity to identify the underlying relationships.

Principal Components Regression (PCR)

Principal Components Regression or PCR is a regression technique that combines PCA and linear regression. PCR responds to multicollinearity by creating a sequence of orthogonal (uncorrelated) principal components of the predictors and using these components in regression. PCR selects the top k components that can explain most of the variance in the data instead of using all of the predictors.

Advantages of PCR:

- It Handles Multicollinearity: By using orthogonal components, PCR can successfully handle multicollinearity of the original predictors.

- Dimensionality Reduction: Since PCR reduces the predictors, it is beneficial for datasets with many variables.
- Focus on Variance: It focuses on components with the most variance.

Limitations of PCR:

- Components Might Not Be So Interpretable: As PCA results in linear combinations of the original variables, they might be less interpretable.
- Variance vs Predictive Power: In PCR, components are chosen based on their variance not their predictive power, and therefore it may lead to less-than-optimal performance if the most predictive variables do not explain most of the variance.
- Modeling, Find Optimal k: A critical step is to choose the number of components (k).

Difference from Other Models:

- Unlike Ridge or LASSO, which operate on the original variables, PCR transforms the predictors into orthogonal components for regression.
- Unlike Regression Trees or SVR, PCR assumes linearity between the transformed predictors and the target variable.

Support Vector Regression (SVR):

Support Vector Regression (SVR) is a regression workhorse of the SVM family. SVR aims to construct a model within a user-set margin of tolerance (epsilon, ϵ), penalizing any data points that exist outside this margin. Using kernel functions, SVR is powerful for nonlinear problems.

Advantages of SVR:

- Good for Non-Linear Relationships: SVR can model complex non-linear relationships using kernel functions.
- Insensitivity to Outliers: SVR optimizes only for points between the margins, making it less sensitive to outliers.
- Controlled Regularization (ϵ): It is a user-specific parameter that gives control over the margin width.

Limitations of SVR:

- Computationally Expensive: The training time increases significantly with large datasets, especially when working with nonlinear kernels.
- Scaling Sensitivity: SVR is sensitive to feature scaling, meaning that the predictors need to be standardized before feeding the SVR model with the data.

Difference from Other Models:

- SVR does not make an assumption about the shape of the relationship between feature and target (like Ridge or LASSO, which assume linearity).
- SVR aims to fit the data within a margin of tolerance (ϵ), as opposed to PCR or Regression Trees minimizing overall error.

Regression Splines:

Regression Splines is a type of piecewise polynomial regression, where data is divided into separate intervals (or knots) and separate polynomial functions are fitted for each interval. The splines are pieced together by ensuring that the pieces meet at the knots where continuity is maintained. Splines can help to model complex non-linear relationships without experiencing overfitting.

Advantages of Regression Splines:

- Flexibility: Captures complex non-linear relationships while maintaining smoothness.

- Local Fitting: Models data locally within each interval, allowing adjustments for changes in the relationship between predictors and response.
- Interpretability: Easier to interpret than other non-linear models such as Regression Trees or SVR.

Limitations of Regression Splines:

- Knot Placement Sensitivity: The selected location and number of knots can have a large impact on model performance.
- Risk of Overfitting: Models with too many knots or high-degree polynomials can lead to overfitting of the data.
- Computational Complexity: Involves careful tuning and choice of knot placement and polynomial order.

Difference from Other Models:

- Unlike Regression Trees that split into disjoint regions, Regression Splines cut the data into multiple segments or regions with smooth transitions between them.
- Regression Splines are more flexible and capable of fitting complex patterns than linear models like Ridge or LASSO.

Regression Tree

A Regression Tree divides data into subsets by selecting splits that minimize a loss function (e.g., mean squared error) in each subset, resulting in a tree-like structure where each leaf node represents a predicted value.

Pros of Regression Trees:

- Easy to Interpret: With its hierarchical tree structure, the algorithm is easy to interpret.
- Non-Linear Relationships: Describes intricate, non-linear relationships between predictors and the response variable.
- Feature Scaling Not Needed: Regression trees are not sensitive to the scale of features.

Drawbacks of Regression Trees:

- Risk of Overfitting: High depth can lead to overfitting; hence, accuracy should be calculated on a separate validation set.
- Instability: A small change in the data could produce a completely different split, leading to instability.
- Less Smooth Predictions: Regression trees can form blocks, resulting in less smooth predictions than Regression Splines or SVR.

Difference from Other Models:

- Regression Trees partition the data into clean, separated sections of the multivariable space, whereas Regression Splines yield smooth transitions.
- Trees can easily cope with non-linear relationships by dividing data into smaller subsets where the relationship becomes linear, unlike linear models (Ridge or LASSO) that must undergo transformations to deal with non-linear data.

Experiment Design

The experiment involved data cleaning/preprocessing, exploratory data analysis (EDA), model building, and metric interpretation. We used Python throughout the project, utilizing multiple packages including Pandas, Sklearn, Matplotlib, Numpy, and Scipy.

Data Cleaning/Exploratory Data Analysis (EDA)

The study began with the cleaning of the datasets in preparation for exploratory data analysis. During the data cleaning missing values were handled (e.g. computed or null values removed), categorical features were encoded, numerical features were scaled, and irrelevant features were removed from the dataset. Exploratory data analysis was critical for familiarization with the dataset (e.g. mean, standard deviation, quantiles, min./max. Values). Summary statistics allowed us to detect and remove outliers that could affect the performance of our models. Any extreme outliers that were identified were removed from the dataset.

```
from sklearn.preprocessing import LabelEncoder

for col in df.columns:
    if df[col].dtype == "object":
        print(str(col))
        label = LabelEncoder()
        label = label.fit(df[col])
        df[col] = label.transform(df[col].astype(str)).astype(int)
df = df.astype({col: 'int' for col in df.select_dtypes(include='object').columns})
```

```
Sex
```

The code above shows the usage of the sklearn.preprocessing labelencoder package. This package is useful when encoding binary categorical variables and ordinal categorical variables such as “sex” or “Freshman”, “Sophomore”, “Junior”, “Senior”. LabelEncoder will transform these binary categories to 0 and 1 (e.g. “Male” = 0, “Female” = 1) and ordinal categories into numerical variables (e.g. “Freshman” = 0, “Sophomore” = 1, “Junior” = 2, “Senior” = 3).

```
df['date'] = pd.to_datetime(df['date'], format="%d/%m/%Y")
df['date'] = df['date'].astype('int64') // 10**9
```

The code above shows the transformation of the “date” column by using the .to_datetie method. This method changes the data type of the date from a string to a numerical format that can be used as a feature in the model.

```
df.drop(['seasons', 'holiday'], axis=1, inplace=True)
df.head()
```

The code above is dropping two columns from the original dataset while not creating a new one. These columns were not relevant to predicting the target variable, so they were dropped so that they wouldn’t hinder the prediction accuracy of the models.

```
df.describe()
```

	rating	num_reviews	price	body	acidity
count	7500.000000	7500.000000	7500.000000	6331.000000	6331.000000
mean	4.254933	451.109067	60.095822	4.158427	2.946612
std	0.118029	723.001856	150.356676	0.583352	0.248202
min	4.200000	25.000000	4.990000	2.000000	1.000000
25%	4.200000	389.000000	18.900000	4.000000	3.000000
50%	4.200000	404.000000	28.530000	4.000000	3.000000
75%	4.200000	415.000000	51.350000	5.000000	3.000000
max	4.900000	32624.000000	3119.080000	5.000000	3.000000

The code above shows the summary statistics for one of the datasets used during this study. This summary can help identify outliers by viewing the max, min, mean, and std. Removing these outliers can prevent the model results being skewed. Also, reviewing the std. Can help identify feature importance. For instance, a feature with a lower standard deviation may be less useful in predicting the target variable.

```
null_counts = df.isnull().sum()
null_counts
```

```
df = df.dropna()
df
```

The last cleaning step featured in this section is the identification and removal of null values. Some machine learning algorithms can't deal with null values and could cause an error when running the code for your model. Missing data can also affect model accuracy and reliability. To avoid errors, it's best to drop null values as shown in the code above or compute the values. Mean imputation is used to compute numerical values, which means that you're replacing missing values with the column's mean value. Mean imputation is best used when data is normally distributed. When data is skewed, it's best to use median imputation, which replaces nulls with the columns median value.

Model Building

After preprocessing our datasets, we moved on to model building which we did in a series of steps. The code for our model before tuning is shown below:

```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import Ridge, Lasso, LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.svm import SVR
```

1. The first step is to import all the necessary packages. Sci-kit learn is used for machine learning models, Numpy is used for mathemical operations, and matplotlib is used for visualizations.

```
X = df.drop(columns=['Rings'])
y = df['Rings']
```

2. Next, we removed the "Ring" column from the main dataset and assigned this column to the variable "y". The rest of the dataset assigned to variable "x" are now independent variables that will be used to predict variable "y", which is the dependent variable.

```
# Normalizing features using StandardScaler
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
```

3. The StandardScalar() object and the .fit_transform method are being used to standardize the dataset, so that all features are equally contributing to the model's prediction. The StandarScalar() object standardizes features by removing the mean and scaling to the unit variance.

```
# Results dictionary to store performance metrics for each model
results = {
    'Regression Tree': {'MSE': [], 'RMSE': [], 'RSE': [], 'RSS': [], 'R^2': []},
    'Ridge Regression': {'MSE': [], 'RMSE': [], 'RSE': [], 'RSS': [], 'R^2': []},
    'Lasso Regression': {'MSE': [], 'RMSE': [], 'RSE': [], 'RSS': [], 'R^2': []},
    'Principal Component Regression (PCR)': {'MSE': [], 'RMSE': [], 'RSE': [], 'RSS': [], 'R^2': []},
    'Regression Splines': {'MSE': [], 'RMSE': [], 'RSE': [], 'RSS': [], 'R^2': []},
    'Support Vector Regression (SVR)': {'MSE': [], 'RMSE': [], 'RSE': [], 'RSS': [], 'R^2': []}
}
```

4. This code shows the creation of a dictionary where each model's metric summary is stored. This format is useful for easy model metric comparison.

```
# Running the models multiple times to compute average metrics
for _ in range(20):
    X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.3, random_state=None)

    # Defining models
    models = {
        'Regression Tree': DecisionTreeRegressor(),
        'Ridge Regression': Ridge(),
        'Lasso Regression': Lasso(),
        'Principal Component Regression (PCR)': make_pipeline(PCA(n_components=num_features), LinearRegression()),
        'Regression Splines': make_pipeline(PolynomialFeatures(degree=3), LinearRegression()),
        'Support Vector Regression (SVR)': SVR()
    }
```

5. The above code shows the usage of the train_test_split function from the sklearn.model_processing package. This function randomly splits the dataset into a training set and a test set. The code above specifies that there should be 20 random splits of the data into training and test sets. The train_test_split function above takes 4 different parameters included the normalized independent variables, the dependent variable, the test size (e.g. the test_size = 0.3 means that 30% of the data will be used in the test set, and the other 70% of the data will be in the training set), and the random_state (e.g. random_state=None establishes that 20 different random splits are done with each iteration). The dataset is split differently with each iteration to reduce bias that can come from using the same split. After splitting the training and test sets, the next step shows each model being defined. The make_pipeline function that is being applied for the Regression Splines and Support Vector Regression models creates a pipeline, or a series of data preprocessing steps for modeling.

```
# Fitting models and computing metrics
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    rss = np.sum((y_test - y_pred) ** 2)
    rse = np.sqrt(rss / (len(y_test) - X_test.shape[1] - 1))
    r2 = r2_score(y_test, y_pred)

    results[model_name]['MSE'].append(mse)
    results[model_name]['RMSE'].append(rmse)
    results[model_name]['RSE'].append(rse)
    results[model_name]['RSS'].append(rss)
    results[model_name]['R^2'].append(r2)
```

6. The code block above shows each model that was defined in the previous code block being “fit” to the training data with the. fit method. “Fitting” the models means that the model is changing its parameters

based on what is learned from the training data. After the model has been fitted, the model is ready to make predictions. The. predict method takes the learned parameters from the training set and applies the model to new data to generate predictions. The chosen metrics are being defined in this code cell, which includes Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Residual Sum of Squares (RSS), Residual Standard Error (RSE), and Coefficient of Determination (R^2).

```
# Define parameter grids for hyperparameter tuning
param_grids = {
    'Ridge Regression': {'ridge__alpha': [0.01, 0.1, 1, 10, 100]},
    'Lasso Regression': {'lasso__alpha': [0.01, 0.1, 1, 10, 100]},
    'Principal Component Regression (PCR)': {'pca__n_components': [1, 2, 3, 4, X.shape[1]]},
    'Regression Splines': {'polynomialfeatures__degree': [2, 3, 4]},
    'Regression Tree': {
      'decisiontreeregressor__max_depth': [3, 4, 5],
      'decisiontreeregressor__min_samples_split': [10, 15, 20],
      'decisiontreeregressor__min_samples_leaf': [4, 8, 10]
},
    'Support Vector Regression (SVR)': {'svr__C': [0.1, 1, 10], 'svr__epsilon': [0.1, 0.5, 1]}
}
```

7. The code block above shows hyperparameter tuning for each model. The hyperparameters are used to manage how the models learn and optimize performance. The dictionary, param_grids, contains the hyperparameters for multiple models. It's important to set these hyperparameters because these settings cannot be learned, they must be set through tuning. For Ridge Regression, the ridge_alpha is being used as the regularization strength. Smaller alpha values (e.g. 0.01) fit the model closer to the data in the training set. Larger alpha values (e.g. 100) are used to apply greater penalties to larger coefficients. A range of alpha values are defined in the code above so that various levels of regularization are tested, which ensures that the optimal alpha value is chosen for the best model fit. Lasso_alpha has a similar function to ridge_alpha, the difference is that Lasso can shrink coefficients to zero. The Principal Component model uses pca_n_components to identify the number of components that will be used during principal component analysis. Principal component analysis is used to reduce the dimensionality of large datasets.
   Regression Splines use `polynomialfeatures_degree` to define the degree of the polynomial used. The higher the polynomial degree, the better the model is at capturing complex relationships, while simultaneously increasing the risk of overfitting. Testing multiple different polynomial degrees also ensures that the model is capable of capturing non-linear relationships.
   Regression Tree uses `max_depth` to limit the depth of the tree, `min_samples_split` to set the minimum number of samples needed to split the node, and `min_samples_leaf` to set the minimum number of samples needed in one leaf. These parameters are necessary to ensure that the model is a good fit for the dataset it's being applied to, reducing the risk of underfitting or overfitting.
   Support Vector Regression uses two hyperparameters, `svr_c` and `svr_epsilon`. `svr_c` is the regularization parameter, and epsilon is the error parameter. Higher values for the regularization parameter (e.g., 10) fit the model closer to the training data, while lower values (e.g., 0.1) increase regularization and simplify the model. With epsilon, higher values produce a larger margin of error, which makes the model less sensitive to variations.

8. GridSearchCV works by going through all possible combinations of the hyperparameters defined in the grid_params dictionary to find the best model. It performs 5-fold cross-validation, meaning that each of the five splits makes up the training data that this model is trained on and validated against to ensure consistency in performance. The metric used for scoring will be the negative mean squared error; this ensures that models with lower errors are better. By setting n_jobs = -1, the process will use all the

available CPU cores to speed up the computations. In the end, the fit function is called to train the model on scaled training data and find an optimal combination of hyperparameters for which the error is minimal. After each model has been tuned, the modeling steps can be repeated, and metrics for the models before tuning and after tuning can be compared to evaluate whether the model has been improved.

Results

Present the experimental results or key findings. Summarize using tables, charts or bullet points.

Lasso Regression

Dataset Name	# of features	Instances	Tuning	MSE	RMSE	RSE	RSS	R^2
taxi_fare	8	206184	No	1.848028	1.358594	1.358682	114311.6	0.999802
taxi_fare	8	206184	Yes	0.00017	0.013046	0.013047	10.52729	1
Health	9	1000	No	35.34829	5.941992	6.02284	10604.49	0.811768
Health	9	1000	Yes	31.0255	5.562219	9307.651	5.637899	0.833247
Daily demand	12	60	No	0.084653	0.235326	0.4465	1.52375	0.99999
Daily demand	12	60	Yes	0.021839	0.147781	0.280395	0.393108	0.999998

Impacts Observed:

- **Feature Reduction:** In a few of the dataset, especially those with larger feature sets, Lasso likely helped in eliminating non-contributive features, simplifying the models.
- **Error Reduction and Model Fit**: There is a clear pattern of reduced errors MSE, RMSE and increased R^2 values post-tuning, indicating better model fit and predictive accuracy.
- **Need for Proper Tuning**: The significant improvements post-tuning highlights the importance of selecting the right alpha value for the penalty term. Proper tuning can dramatically enhance model performance.

The models created throughout this project suggest that Lasso regression, when correctly tuned, is highly effective for predictive tasks across various domains, making it a versatile tool for both simplifying the models and improving their predictive accuracy.

Ridge Regression

Dataset Name	# of features	Instances	Tuning	MSE	RMSE	RSE	RSS	R^2
Auto_MPG	7	378	Yes	11.45923	3.37326	3.507348	1375.107	0.815125
Auto_MPG	7	378	No	9.300952	3.049746	3.170975	1116.114	0.837938
Fuel Consumption	10	22556	Yes	0.006646	0.081497	0.08157	44.97551	0.993367
Fuel Consumption	10	22556	No	0.006577	0.081096	0.081168	44.50402	0.993256

Car Information	9	300	Yes	0.193935	0.439739	0.453393	22.88433	0.803637
Car Information	9	300	No	0.168361	0.410318	0.423058	19.86659	0.806633

Impact Observed:

- **Inconsistent Tuning Results**: Unlike Lasso, the results after tuning in Ridge regression doesn't consistently show improvement. This could indicate challenges in finding the optimal alpha that properly penalizes the coefficients without losing significant predictive power.

- **Minimal Impact on Large Datasets**: For the "Fuel Consumption" dataset, the impact of Ridge regression, both pre and post-tuning, is minimal, indicating that the dataset might be robust enough or that the features are already well-conditioned.

- **Potential Over-Penalization**: In datasets like "Auto_MPG" and "Car Information," the tuning process seems to over-penalize the model, as seen from the reduced R^2 values post-tuning. This might suggest we need to revisit the tuning process to find a balance that doesn't reduce the model performance.

Results gathered throughout this project, we see the subtle effects of Ridge regression, which may vary depending on the dataset characteristics and the tuning of hyperparameters. It's crucial to adjust the regularization parameter carefully to avoid underfitting and ensure the model maintains its ability to generalize well from the training data to unseen data.

Principal Component Regression

Dataset Name	# of features	Instances	Tuning	MSE	RMSE	RSE	RSS	R^2
Auto_MPG	7	378	Yes	11.48586	3.377341	3.511591	1378.304	0.814677
Auto_MPG	7	378	No	9.271774	3.044959	3.165997	1112.613	0.838447
Health	9	1000	Yes	31.1515	5.576015	5.651882	9345.449	0.834142
Health	9	1000	No	31.01689	5.56141	9305.066	5.637079	0.833289
Car Information	13	22556	Yes	0.006641	0.081465	0.081537	44.93997	0.993372
Car Information	13	22556	No	0.006577	0.081097	0.081169	44.50488	0.993256

Impact Observed:

- **Dimensionality Reduction Impact**: For datasets like "Car Information," the slight improvements post-tuning suggest that PCR is effectively capturing essential features while reducing noise.

- **Possible Overfitting or Information Loss**: In the "Auto MPG" dataset, the deterioration in model performance after tuning could indicate that not all relevant variance is being captured, possibly due to discarding components that contain useful predictors.

- **Marginal Improvements**: In cases like the "Health" dataset, the changes are very minimal, suggesting that PCR might be neutral in terms of its benefits, possibly because the original features are already effective predictors.

Results gathered throughout this creating these models suggest that while PCR can simplify models and potentially improve performance by focusing on key variances, the tuning of how many components to retain is critical. Incorrectly specifying the number of principal components can lead to essential information being lost or unnecessary complexity being added, impacting model accuracy and generalizability.

Regression Splines

Dataset Name	# of features	Instances	Tuning	MSE	RMSE	RSE	RSS	R^2
Student Performance	20	6607	Yes	0.435674	0.657803	0.659802	863.94158	0.5713
Student Performance	20	6607	No	0.323394	0.568678	0.570406	641.29110	0.64381
Health	9	4002	Yes	0.116407	0.341082	0.342368	139.68868	0.534057
Health	9	4002	No	0.11085	0.332942	0.334198	133.02042	0.556538
Taxi Fare	8	206184	Yes	6.43E-26	2.32E-13	2.32E-13	3.98E-21	1
Taxi Fare	8	206184	No	2.19E-26	1.48E-13	1.48E-13	1.35E-21	1

Impact Observed:

- **Sensitivity to Tuning**: The results show a general trend where tuning does not necessarily improve and often worsens the model performance for the "Student Performance" and "Health" datasets. This could imply overfitting or misconfiguration in how the knots are placed or how many are used.

- **Need for Careful Knot Placement**: Proper placement and number of knots are crucial in spline regression to balance the model's complexity and its ability to generalize. Poorly placed knots can lead to overfitting or underfitting, which can be deduced from the decreased performance metrics post-tuning.

- **Assessment of Fit Quality**: The "Taxi Fare" results suggest a perfect fit, which could either mean an extremely well-tuned model or potential issues like overfitting during model training.

In practice, regression splines require careful consideration of the underlying data structure and the specific non-linear trends present. Effective tuning, including the proper choice and number of knots, is important for optimizing the performance and predictive power of spline-based models.

Regression Tree

Dataset Name	# of features	Instances	Tuning	MSE	RMSE	RSE	RSS	R^2
Student Performance	14	45001	Yes	0.104104	0.322628	0.322795	1405.4	0.397828
Student Performance	14	45001	No	0.066798	0.258454	0.258588	901.7762	0.614167
Wine Quality	11	7500	Yes	0.004863	0.069685	0.069869	9.2355	0.687097
Wine Quality	11	7500	No	0.003582	0.05985	0.060008	6.802184	0.788991
Seoul Biking Sharing Demand	13	8760	Yes	0.222033	0.471025	0.472104	583.5016	0.779368
Seoul Biking Sharing Demand	13	8760	No	0.295901	0.543968	0.545214	777.6269	0.699634

Impact Observed:

- **Overfitting Risk**: Tuning regression trees involves setting parameters like the maximum depth, minimum samples per leaf, minimum samples per split, and possibly pruning. Incorrect settings can lead to overfitting, that is evident in the "Student Performance" and "Wine Quality" datasets where performance decreased post-tuning.

- **Underfitting Correction**: Effective tuning can correct underfitting by allowing the tree to capture complex patterns, seen in the "Seoul Biking Sharing Demand" dataset.

- **Model Complexity Management**: The balance between a tree's depth and its pruning level is important. Overly complex trees might fit the noise in the training data, whereas overly simplistic trees might not capture sufficient data nuances.

The results gathered highlight the sensitivity of regression trees to their hyperparameter settings and the importance of careful tuning to balance the model complexity against its ability to generalize. Each dataset's characteristics can significantly influence how tuning impacts model performance, outlining the need for dataset-specific model evaluation and adjustments.

Support Vector Regression

Dataset Name	# of features	Instances	Tuning	MSE	RMSE	RSE	RSS	R^2
Health Insurance Charge	7	1338	Yes	0.15093	0.387452	0.390376	60.6739	0.844237
Health Insurance Charge	7	1338	No	0.136711	0.369745	0.372535	54.95787	0.863365
Diabetes	34	70000	Yes	2.688692	101.0326	101.2723	60466090	0.01319
Diabetes	34	70000	No	1.817769	1.639723	1.641052	56462.53	0.80687
Apple Quality	13	8760	Yes	0.087025	0.294844	0.295956	104.4298	0.651661
Apple Quality	13	8760	No	0.083807	0.289495	0.290586	100.5686	0.664726

Impact Observed:

- **Tuning Sensitivity**: SVR is sensitive to its hyperparameters, the kernel type (linear, polynomial, RBF, etc.), and the epsilon in the loss function. Incorrect settings can lead to poor performance, particularly in datasets with high complexity or large feature sets.
- **Potential Overfitting**: Especially in the case of the Diabetes dataset, poor tuning can lead to drastic overfitting, where the model excessively captures noise instead of the underlying data pattern.
- **Balancing Complexity**: For datasets with many features, like Diabetes, managing the complexity of the model through proper kernel selection and parameter setting is crucial to ensure that the model does not just memorize the training data.

These results pinpoint the importance of careful hyperparameter selection and validation when using SVR. Properly tuned, SVR can be a powerful regression tool, capable of effectively handling non-linear relationships and high-dimensional spaces. However, missteps in tuning can lead to a significantly poor-performing model, emphasizing the need for thorough cross-validation and parameter exploration.

Conclusion

This project was a comprehensive evaluation of six different regression models Ridge Regression, Lasso Regression, Principal Component Regression (PCR), Regression Splines, and Support Vector Regression (SVR) across 20 diverse datasets. Initial evaluations without hyperparameter tuning established baseline performances, which were significantly enhanced through systematic hyperparameter tuning, demonstrating the importance of this process in optimizing model accuracy. Models such as Lasso and Ridge were particularly effective, highlighting the value of regularization in managing complexity and preventing overfitting. This project adeptly managed computational and time constraints, establishing best practices for model selection and hyperparameter tuning. The findings contribute valuable insights into regression techniques' handling of data complexities and pave the way for future explorations in applying these models to new datasets and expanding the range of regression models used, enhancing both practical applications and theoretical understanding within the field of data science.

References

Include all references (books, articles, webpage links, etc.) that you cited.

https://scikit-learn.org/1.5/modules/generated/sklearn.preprocessing.StandardScaler.html

Appendices

Datasets:

1.Song Popularity Dataset:
https://www.kaggle.com/datasets/yasserh/song-popularity-dataset

2.Seoul Bike Sharing Demand:
https://archive.ics.uci.edu/dataset/560/seoul+bike+sharing+demand

3.Health Insurance charges:
https://www.kaggle.com/datasets/teertha/ushealthinsurancedataset/data

4.Student Performance Factors:
https://www.kaggle.com/datasets/lainguyn123/student-performance-factors

5.Gym Membership Dataset:
https://www.kaggle.com/datasets/ka66ledata/gym-membership-dataset

6.Diabetes Dataset:
https://www.kaggle.com/datasets/ankitbatra1210/diabetes-dataset

7.Health
https://www.kaggle.com/datasets/pratikyuvrajchougule/health-and-lifestyle-data-for-regression

8.Appliances Energy
https://archive.ics.uci.edu/dataset/374/appliances+energy+prediction

9.Apple Quality:
https://www.kaggle.com/datasets/nelgiriyewithana/apple-qualityLo

10. Loan Approval
https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data

11. Student Health and Attendance:
https://www.kaggle.com/datasets/ziya07/student-health-and-attendance-data

12. Apartment rent classified:
https://archive.ics.uci.edu/dataset/555/apartment+for+rent+classified

13. Auto MPG:
https://archive.ics.uci.edu/dataset/9/auto+mpg

14. Taxi trip fare prediction:
https://www.kaggle.com/datasets/raviiloveyou/predict-taxi-fare-with-a-bigquery-ml-forecasting?resource=download

15. Spanish Wine Quality :
https://www.kaggle.com/datasets/fedesoriano/spanish-wine-quality-dataset/data

16. Predicting the age of abalone from physical measurements:
https://www.kaggle.com/datasets/rodolfomendes/abalone-dataset/data

17. Fuel Consumption Prediction:
https://www.kaggle.com/datasets/ahmettyilmazz/fuel-consumption/data

18. Advertising Dataset:
https://www.kaggle.com/datasets/ashydv/advertising-dataset/data

19. Car information dataset:
https://www.kaggle.com/datasets/tawfikelmetwally/automobile-dataset/data

20. Daily Demand Forecasting Orders:
https://archive.ics.uci.edu/dataset/409/daily+demand+forecasting+orders