Application and Comparison of Deep Neural Networks for Steel Quality Prediction in Continuous Casting Plants

Project in Machine Learning (190.015), SS2023

Zeinab Naji[1]

[1] *zeinab.naji@k1-met.com*, *Montanuniversität Leoben, Austria*

January 2, 2024

Machine learning algorithms detect natural data patterns, providing insights for improved decision-making and predictions. In this project three different supervised ML models including linear regression, polynomial regression, and support vector machine are used to predict steel quality from the continuous casting plants dataset provided. First, models are trained to learn and map the input process data to the target steel quality variables. Then, ML architectures are compared to identify the most effective model for accurate steel quality prediction. Finally, the performance of implemented models are assessed using appropriate evaluation metrics.

1 Introduction

Machine learning (ML) algorithms excel at identifying patterns and relationships within large and complex datasets that might not be immediately apparent to humans. This ability allows for better understanding and utilization of data. By analyzing data comprehensively, ML assists in making data-driven decisions. These decisions are often more accurate and can lead to better outcomes across different domains. ML models can predict future trends or outcomes based on historical data patterns. This capability is valuable across various fields, including industry, finance, healthcare, weather forecasting, and more. Overall, the importance of using ML techniques lies in their capability to analyze vast amounts of data, extract valuable insights, and facilitate informed decision-making, which can significantly impact efficiency, innovation, and problem-solving across industries.

In this project, a significant volume of steel production datasets, comprising recorded sensory values and process data from 'Stahl- und Walzwerk Marienhütte GmbH Graz,' is initially prepared and preprocessed for ML tasks. Subsequently, these datasets are utilized in employing the linear regression method, polynomial regression method, and support vector machine model. The primary goal is to identify the most effective algorithm capable of capturing underlying data patterns and predicting a quality-relevant metric, specifically the yield strength.

The content of this report is as follows: In Section 2, the data preparation methods are described, including accessing and preprocessing the data, data visualization with histograms, pairplots, scatter plots, and correlation matrix, as well as outlier detection and data cleaning. In section 3, the results of using various types of ML models for predictive analysis and their performance on the dataset are investigated. Comparion between different methods and conclusion are discussed in Section 4.

2 Data preparation methods

Data preparation is a foundational step in ML, involving the cleaning, transformation, and formatting of raw data to ensure its suitability for training models. It ensures that high-quality, relevant data is available for effective algorithm training and accurate predictive outcomes.

2.1 Access and preprocess the data

The normalized training and testing datasets are initially loaded using the `Pandas` library in Python. Subsequently, they are inspected to obtain summary statistics such as shape, size, and values. The training and testing datasets have shapes of (7642, 22) and (3337, 22) respectively. The first value indicates the number of rows, while the second value refers to the number of columns. Both datasets consist of 21 columns for inputs labeled 'input1' through 'input21', and one column for output labeled 'output'. All values fall within the range of float numbers between 0.0 and 1.0.

Afterwards, the datasets are checked for null or missing values using the `.isnull().sum()` method. No missing or null data is found in either dataset.

Furthermore, training and testing datasets are checked for any duplicated rows or columns, and no duplication is found. Additionally, there are no non-numeric values present in the dataset.

2.2 Data visualization

Visualizations are essential in exploring data distributions and relationships, facilitating the identification of patterns and insights.

In this study various types of plots such as heatmaps, pair plots, histograms, scatter plots, and normal probability plots are employed to explore data's characteristics and relationships between variables.

2.2.1 Histograms

Histograms are graphical representations that display the distribution of numerical data. They consist of a series of contiguous rectangular bars, where the width of each bar represents a range of values, and the height represents the frequency or count of occurrences within that range. They help in understanding the distribution of data, and identification of patterns, skewness, potential outliers and etc. within the dataset.

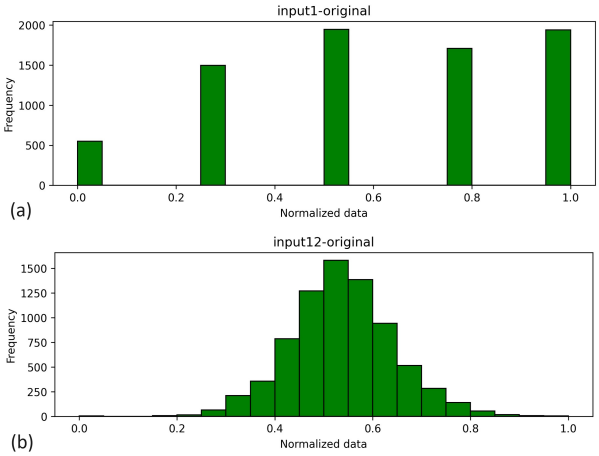


Figure 1: *Histograms of (a) the input1 and (b) the input12.*

Although histograms for all the data in the training dataset have been created throughout this study, this report presents only the histograms for two specific inputs, input1 and input12, shown in figure 1. It can be seen that while the datapoints in input1 are distributed across five bins, signifying an uneven distribution among these categories, input12 exhibits zero skewness and displays a symmetrical distribution ranging between 0.0 and 1.0. To create plots a Python class called `DataVisualizer` is developed which is specifically for visualizing data. A function called `hist_all` is defined to generate histograms and below main part of it is shown.

```
# ****************************
# The hist_all Function
class DataVisualizer:
    def __init__(self,
        train_data_path='
        normalized_train_data.csv',
        cleaned_train_data_path='
        cleaned_train_data.csv')
    def hist_all(self, x=[2,3,4]):
        for i, j in enumerate(x):
            axs[i, 0].hist(self.
                train_data['input%d'
                %j], bins=20, color=
                'green', edgecolor='
                black')
            axs[i, 1].hist(self.
                train_data_c['input%
                d'%j], bins=20,
                color='pink',
```

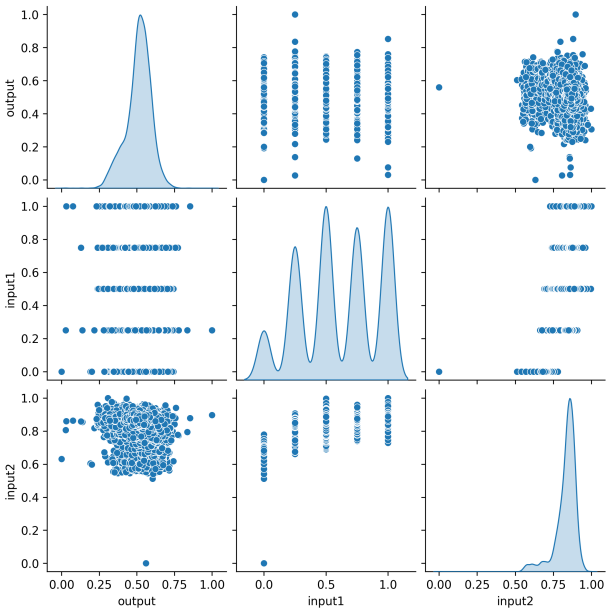


Figure 2: *Pairplot of the output, input1 and input2.*

```
            edgecolor='black')
        plt.show()
        pass
```

2.2.2 Pairplots

A pairplot is a type of data visualization technique commonly used in exploratory data analysis (EDA) to visualize relationships between pairs of variables in a dataset. It is a grid of plots where each variable in the dataset is paired with every other variable, creating a matrix of scatterplots for numerical variables and histograms for the diagonal representing distributions of individual variables.

Figure 2 displays a 3 by 3 pairplot (as a part of 22 by 22 pairplot) generated using Python's `Seaborn` library, which shows correlations among the output, input1 and input2. Notably, input2 exhibits a distinct cluster in correlation with the output, contrasting input1 and the output, as well as the relationship between input1 and input2. This visualization helps investigate correlations and also outliers within the dataset.

2.2.3 Scatter plots

Alongside the pairplots used for initial analysis, detailed scatter plots are created to further explore the connections between variables in the dataset. In figure 3 scatter plots depicting the relationships between input11 and input12 variables against the output are shown. These scatter plots are effective tools for thoroughly examining the data, facilitating not only detailed exploration but also the identification of potential outliers within the dataset. To generate scatter plots of the output vs. any input dimension a Python function called `scatterplt` is developed within the `DataVisualizer` class which takes a list of inputs as arguments. the code below is part of the mentioned function to create scatter plots.

```
# ****************************
# The scatterplt Function
class DataVisualizer:
    def __init__(self,
        train_data_path='
        normalized_train_data.csv')
    def scatterplt(self, x=[2,3,4])
        :
        y_train = self.train_data['
            output']
        plt.scatter(self.train_data
            [f'input{j}'], y_train,
            color=random.choice(self
            .colors))
        plt.show()
        pass
```

2.2.4 Correlation matrix

A correlation matrix is a table that shows the correlation coefficients between many variables. Each cell in the table displays the correlation between two variables. The most common correlation coefficient used in correlation matrices is the Pearson correlation coefficient, which measures the linear relationship between two variables. The formula for calculating the Pearson correlation coefficient, r, between variables X and Y from a dataset is:

$$r = \frac{\sum (X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum (X - \bar{X})^2 \sum (Y - \bar{Y})^2}}, \qquad (1)$$

where $\bar{X}$ and $\bar{Y}$ are the means of variables X and Y, respectively. The correlation coefficient, which falls between -1 and 1, quantifies both the strength and direction of a linear relationship between variables.

Figure 4 illustrates the correlation matrix (heatmap) between the output and the initial 10 input variables. A value of 1, represented by the deepest green shade, indicates a perfect positive linear relationship. Conversely, -1, shown as the

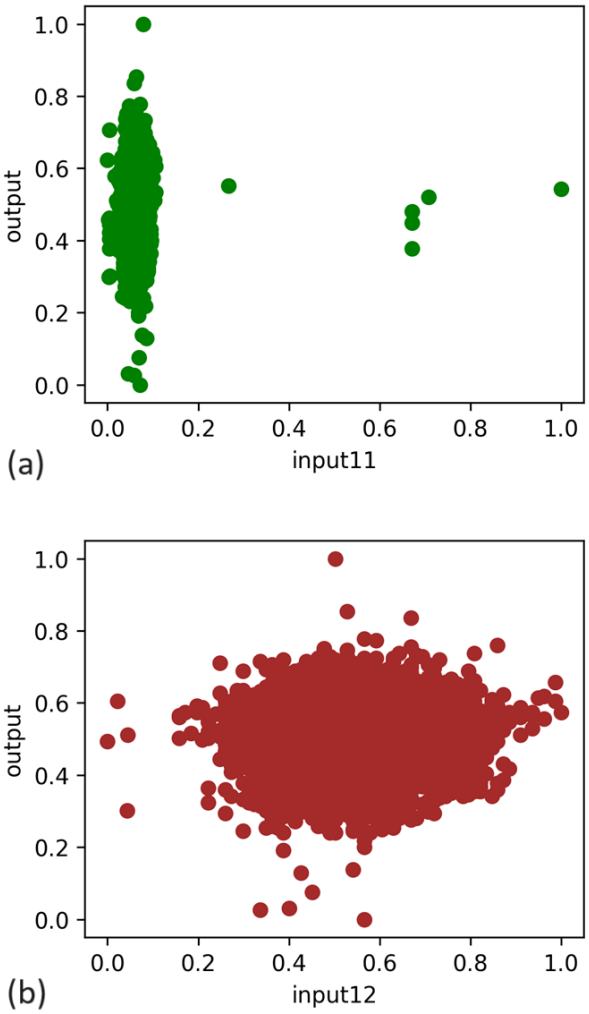


Figure 3: *Scatter plots of (a) input11 against the output, and (b) input12 against the output.*

deepest pink shade, signifies a perfect negative linear relationship. A value of 0 represents no linear association between the variables. In this figure, input1 and input2 exhibit the strongest positive relationship ($r = 0.63$), whereas input1 and input6 display the strongest negative relationship ($r = -0.96$).

In general, heatmaps are useful because they provide visual representations of complex data patterns, allowing for quick interpretation and identification of relationships or variations within the dataset.

2.3 Outlier detection and data cleaning

Data cleaning is a critical step in the preprocessing phase of any analytical study, especially in ML and statistical analysis. Outliers, data points that significantly differ from the rest of the dataset, can significantly impact the accuracy and reliability

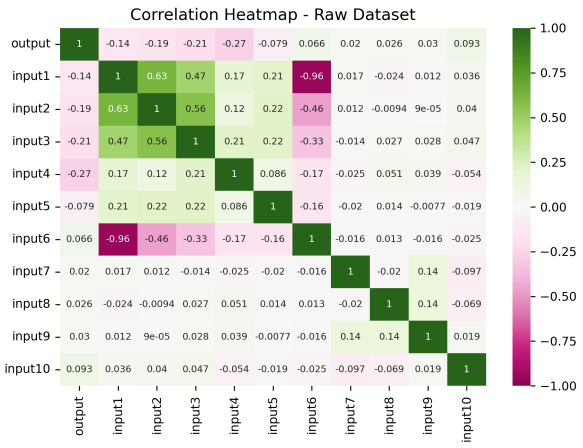


Figure 4: *Correlation heatmap of the output and the first 10 input varuables.*

of predictive models. Various methods exist to detect and handle outliers, among which the Z-score method stands out for its effectiveness. This study presents a comparative analysis showcasing the impact of applying the Z-score method and replacing outliers with the median on the distribution of data, visualized through normal probability plots and histograms. By examining these plots before and after the cleaning process, we aim to demonstrate the efficacy of this technique in improving data quality for subsequent analysis.

2.3.1 Normal probability plots

Within statistical analysis, various models, such as linear regression, rely on the assumption of data being normally distributed. Therefore, checking the normality assumption is crucial to ensure the validity of these analyses. Normal probability plots provide a visual way to assess this assumption.

In this project, normal probability plots are generated for all 21 inputs and the output in the training dataset, followed by the calculation of the coefficient of determination (R^2) to evaluate the goodness of fit to theoretical distributions.

Subsequently, the Z-score method is employed to identify outliers or extreme values within the dataset. This statistical technique standardizes and measures the distance of a data point from the dataset's mean in terms of standard deviations.

Following outlier identification, the dataset undergoies a replacement process inwhich outliers are substituted with the median value. This cleaned dataset is then stored in a new .csv file, referred to as 'cleaned data' from this point forward.

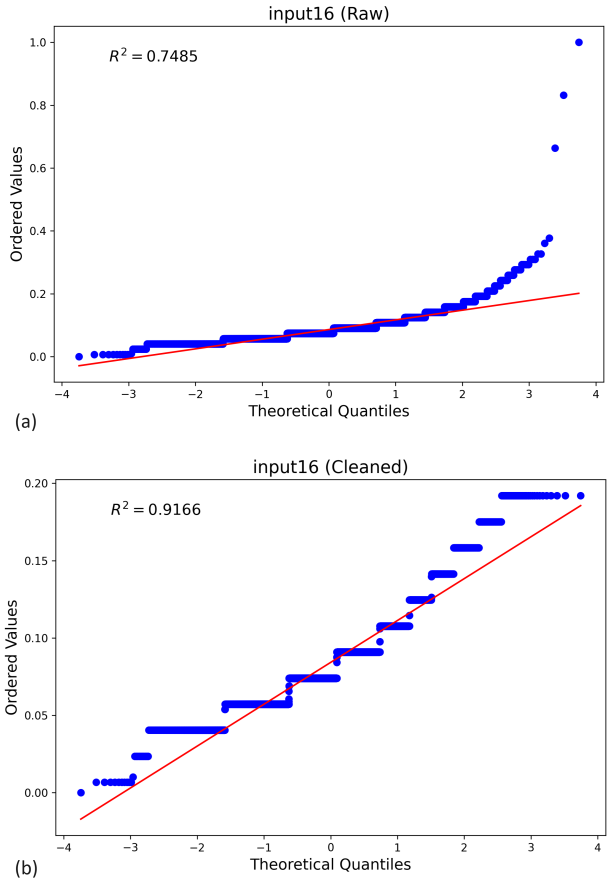


Figure 5: *Normal probability plots of the variable input16 from trained dataset (a) with raw data and (b) cleaned data.*

To assess the impact of the replacement process, normal probability plots are generated again using the cleaned data. As a result, 44 subplots are generated for the raw data (before replacement) and the cleaned data (after replacement) overall.

Normal probability plot of the variable input16 from the trained dataset is shown in figure 5. This plot is created using `scipy` and `matplotlib` libraries in Python. The x-axis represents the theoretical quantiles of a standard normal distribution, and the y-axis shows the ordered values from the dataset being analyzed. It can be seen that outliers have been replaced, and the value of R^2 has increased from 0.7485 to 0.9166. It is good to mention that R^2 ranges from 0 to 1, and values closer to 1 suggest a better fit to the normal distribution, while values closer to 0 indicate a poorer fit.

Furthermore, histograms of the same variable before and after cleaning are illustrated in figure 6. Figure 6a, which shows the histogram prior to outlier removal, reveales a distribution with

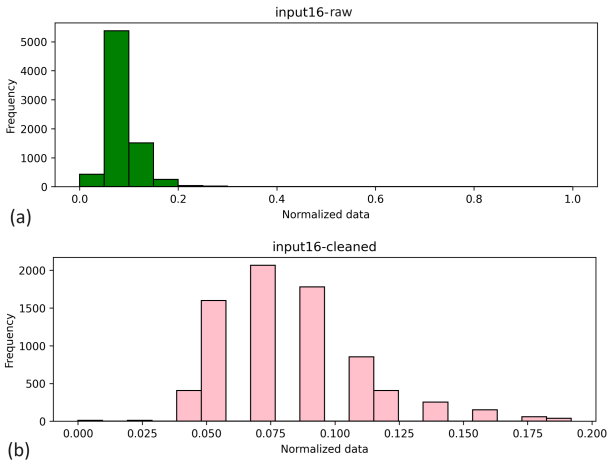


Figure 6: *Histograms of the variable input16 from trained dataset (a) with raw data and (b) cleaned data.*

only six bins between the values of 0.0 and 0.3. The highest frequency (more than 5000), indicates a concentrated peak within this interval. Figure 6b which is subsequent to outlier treatment and replacement with the median, displayes a more detailed distribution with 12 bins within the same range. The maximum frequency is more than 2000, signifying a reduction in peak intensity but revealing a finer spread of data points within this range.

In general, outliers can elongate tails or create extra peaks, impacting the normal representation of data distribution, and removing outliers can result in more consistent bin heights, leading to a more even and smoother histogram appearance.

3 Machine learning models

In this study, based on the problem type and the nature of the datasets, which involve regression analysis, three ML methods of linear regression, polynomial regression, and SVM have been employed.

In all methods the 'cleaned data' is used for training. Since a separate dataset exists for testing purposes, there is no requirement for data splitting. To maintain brevity in this report, focus has been placed on a subset of variables: 'output', 'input12', 'input6', and 'input21', which are utilized and discussed in detail.

The evaluation of models is conducted using common regression performance metrics including the mean squared error (MSE), root mean

squared error (RMSE), and standard deviation of residuals (STD). These metrics provide insights into the models' predictive accuracy and error distribution.

Hyperparameter tuning techniques, like grid search, are utilized to optimize the models' performance by refining and adjusting their parameters.

Finally, the models are deployed to make predictions on the testing dataset, allowing for a practical assessment of their predictive capabilities.

3.1 Linear regression model

To implement a linear regression model, first, a regression model is created using the `LinearRegression` class from the scikit-learn (`sklearn`) library in Python. Subsequently, the linear model is fitted to the dataset. Once the model is trained, it is utilized for predictions using the test dataset. Following this, errors are calculated using the functions `mean_squared_error` and `r2_score` from `metrics` class. Afterwards, a scatter plot of the trained data is generated, and the fitted regression line is added to the plot. The main parts of the algorithm are as follows.

```
# ****************************
# Algorithm of Linear Regression
from sklearn.linear_model import
    LinearRegression
from sklearn.metrics import
    mean_squared_error, r2_score
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test,
    y_Line)
rmse = mean_squared_error(y_test,
    y_Line, squared=False)
residuals = y_test - y_Line
std_deviation = np.std(residuals)
slope = model.coef_[0]
intercept = model.intercept_
```

To perform a search for the best-fitted line, coefficients of the fitted line (slope and intercept) are obtained. Both values are then incremented and decremented by 0.1, and new lines using the line formula

$$y = \text{slope} \times x + \text{intercept} \tag{2}$$

are created. As a result, three values for the slope and three values for the intercept yield a total of nine fitted lines.

Figure 7 compares nine different linear regression models using variables input12 and input21, and table 1 presents the error values corresponding to these models. Figure 7e displays the predicted linear regression line generated by the model itself, while the remaining plots represent variations resulting from adjustments in slope and intercept. Althogh figure 7e is considered as the best fit among other models, it exhibits a higher MSE compared to certain alternative models. However, a lower MSE alone might not always indicate the best model as it solely measures the average squared deviation between predicted and observed values. A model with higher error, yet considered the best fit, signifies a deep grasp of complex data, outperforming simpler models. This elevated error might stem from the model's ability to capture detailed patterns and handle outliers. It may also balance accuracy and avoiding overfitting, leading to better predictions on new, unseen data.

Meanwhile, the application of the linear regression model to analyze the relationship between the independent variable, input6, and the dependent variable ,output, is illustrated in figure 8.

3.2 Polynomial regression model

To apply a polynomial regression model to the dataset, after determination of the desired polynomial degree, the `PolynomialFeatures` class from the scikit-learn library in Python is used to transform the original features into polynomial features, generating higher-order terms and interactions based on the specified degree. For instance, if the original features are x and y, transforming them into polynomial features might involve creating new variables like x^2, y^2, xy, etc., up to a defined degree (e.g., second degree, third degree, etc.). The subsequent implementation steps largely resemble those of linear regression, previously discussed, though abbreviated here for brevity. Importatnt parts of the algorithm are outlined below.

```
# ****************************
# Algorithm of Polynomial
    Regression
from sklearn.linear_model import
    LinearRegression
from sklearn.preprocessing import
    PolynomialFeatures
for i in [2,3,4]:
    degree = i
```

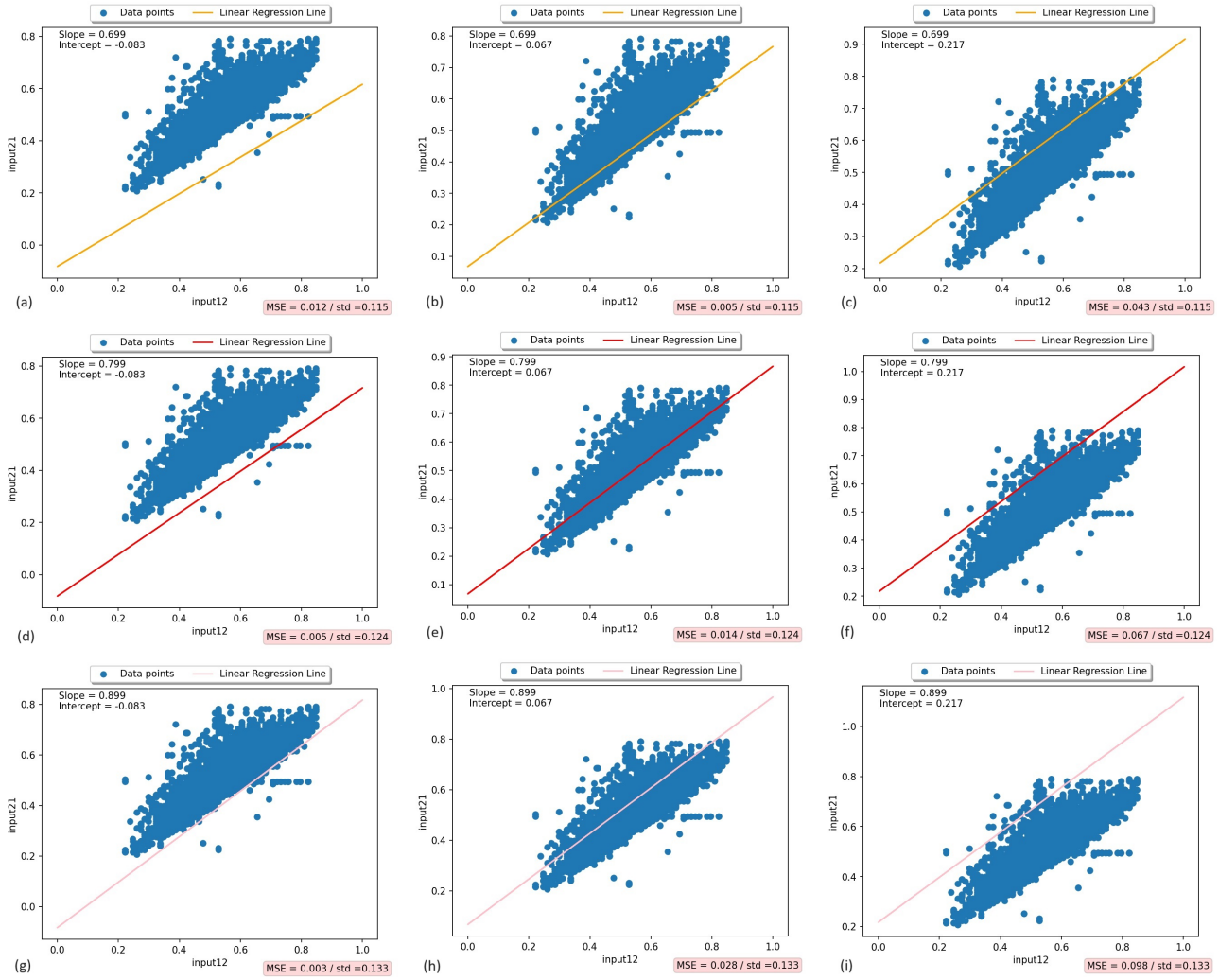


Figure 7: *Comparison of nine linear regression models for input12 vs. input21.*

```
poly = PolynomialFeatures(
    degree=degree)
X_train_poly = poly.
    fit_transform(X_train)
X_test_poly = poly.transform(
    X_test)
model = LinearRegression().fit(
    X_train_poly, y_train)
y_pred = model.predict(
    X_test_poly)
```

In this study, degree of the polynomial is considered as a hyperparameter. Therefore, to determine the optimal degree of the polynomial that best fits the data, efforts are directed towards tuning this hyperparameter. Figure 9 presents a comparison between three polynomial regression models with degrees 2, 3, and 4 using variables of input12 and input21, while the corresponding error values are tabulated in table 2. Upon closer examination of the table, it is evident that figure 9b exhibits the lowest MSE and STD. Additionally, figure 10 depicts a similar comparison using variables of input6 and the output. The corresponding error values are summarized in table 3. Notably, the minimum MSE and STD are observed for Figure 10b. Moreover, visually inspecting figure 9b and figure 10b reveals that these particular models not only demonstrate the lowest error values but also capture the underlying detailed patterns most effectively. Consequently, among the models considered, the polynomial regression model with degree 3 stands out as the most favorable.

3.3 Support vector machine

The final ML method which is used to model and analyze the relationship between variables is SVM.

To apply this method, first, a support vector regression (SVR) model is constructed with a radial basis function (RBF) kernel, which offers flexibil-

Table 1: *Error metrics of linear regression models for input12 vs. input21.*

Plot	Slope	Intercept	MSE	RMSE	STD
a	0.699362	-0.082779	0.011823	0.108735	0.115105
b	0.699362	0.067221	0.004862	0.069728	0.115105
c	0.699362	0.217221	0.042901	0.207125	0.115105
d	0.799362	-0.082779	0.00473	0.068773	0.123727
e	0.799362	0.067221	0.013567	0.116477	0.123727
f	0.799362	0.217221	0.067404	0.259623	0.123727
g	0.899362	-0.082779	0.003458	0.058801	0.132825
h	0.899362	0.067221	0.028093	0.16761	0.132825
i	0.899362	0.217221	0.097729	0.312616	0.132825

Table 2: *Error metrics of polynomial regression models for input12 vs. input21.*

Plot	Degree	MSE	RMSE	STD
a	2	0.013452	0.115982	0.051673
b	3	0.013383	0.115683	0.051051
c	4	0.013612	0.116669	0.051464

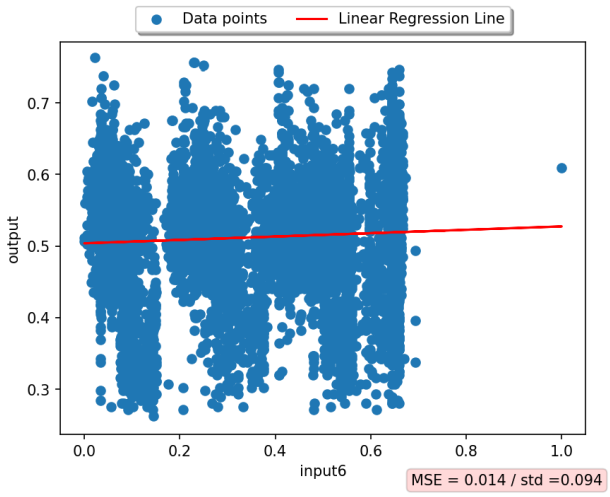


Figure 8: *Linear regression model of input6 vs. the output.*

ity in handling non-linear relationships. For this purpose the `SVR` class from the scikit-learn library is utilized.

Next, hyperparameters, C, gamma, and epsilon, are defined and then optimized through a grid search technique. This process is implemented using `GridSearchCV` class from scikit-learn library in Python. A grid of specified parameter values ([0.1, 1.0, 10.0] for C, [0.1, 0.5, 1.0] for gamma, and [0.1, 0.2, 0.3] for epsilon) is explored to find the combination that minimizes MSE during cross-validation (cv). In the developed code the value of 3 is considered for cv, and it means the data will be divided into 3 equal parts (or folds). The model will be trained and evaluated three times, each time using a different fold as the validation set and the remaining folds for training. Cross-validation helps in estimating how well a model will perform on unseen data and provides a more reliable evaluation of the model's performance compared to a single train/test split. It reduces the risk of overfitting or underfitting by using multiple subsets of the data for both training and validation.

After fitting the grid search object to the training data, the best hyperparameters and the corresponding best model are obtained using `best_params_` and `best_estimator_` attributes, respectively. In the next step the best model is utilized to make predictions on the test data.

For output and visualiziation, the code generates a table displaying best hyperparameters, and scatter plots comparing the predicted vs. actual

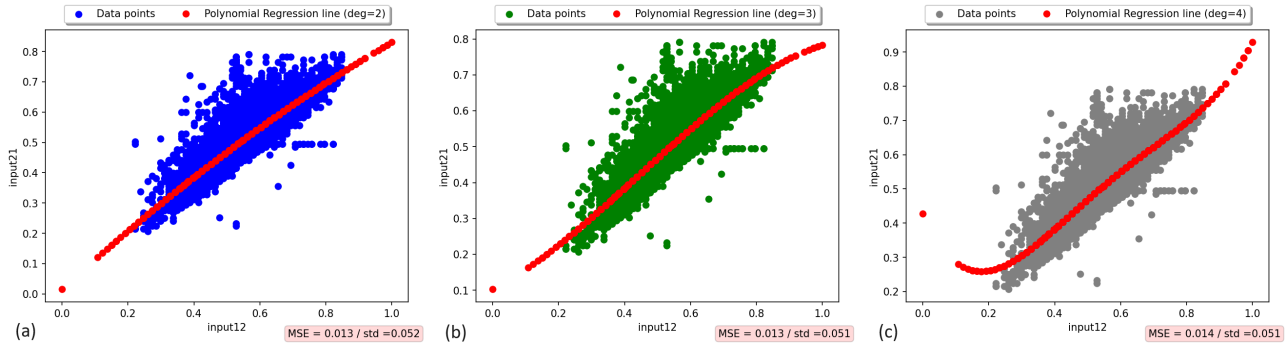


Figure 9: *Comparison of three polynomial regression models with different degrees of 2, 3 and 4 for input12 against input21.*

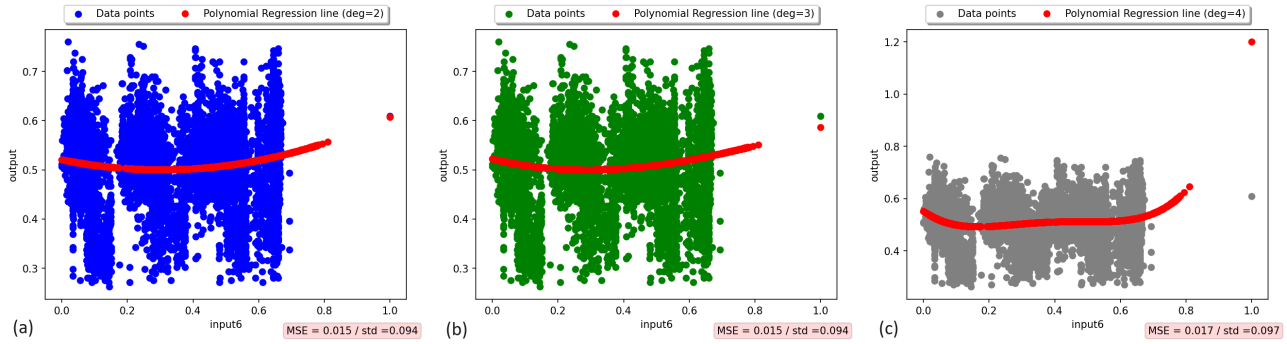


Figure 10: *Comparison of three polynomial regression models with different degrees of 2, 3 and 4 for input6 against the output.*

values for evaluation and further analysis. Figure 11 displays the optimal SVM models with corresponding best hyperparameters based on minimum MSE for two sets of variables, input12 vs. input 21 and input6 vs. the output. The values of the attribute `mean_test_score`, given in the tables, indicate the performance of the model using each specific combination of hyperparameters during cross-validation, which aid in the selection of the best hyperparameters for the model. In this case, the maximum value of this performance metric indirectly minimizes the MSE. Key commands of the implemented code are highlighted in the following section.

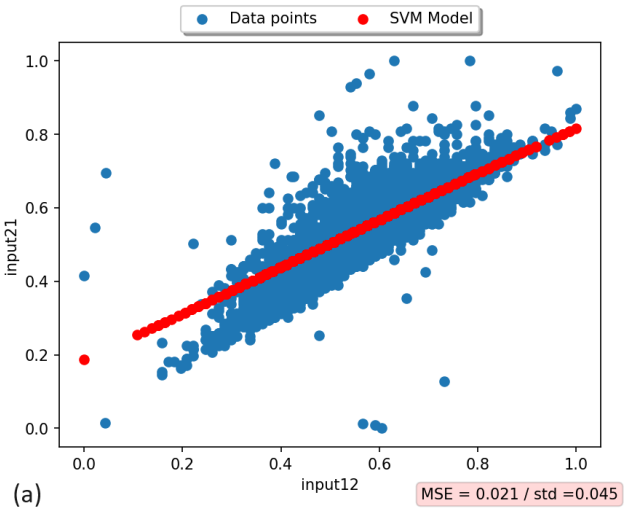
```
# ******************************
# Algorithm of SVM
from sklearn.svm import SVR
from sklearn.model_selection import
    GridSearchCV
param_grid = {
    'C': [0.1, 1.0, 10.0],
    'gamma': [0.1, 0.5, 1.0],
    'epsilon': [0.1, 0.2, 0.3]
}
svr = SVR(kernel='rbf')
grid_search = GridSearchCV(
    estimator=svr, param_grid=
    param_grid,cv=3, scoring='
    neg_mean_squared_error')
grid_search.fit(X_train, y_train)
best_params = grid_search.
    best_params_
best_model = grid_search.
    best_estimator_
y_pred_best = best_model.predict(
    X_test)
```

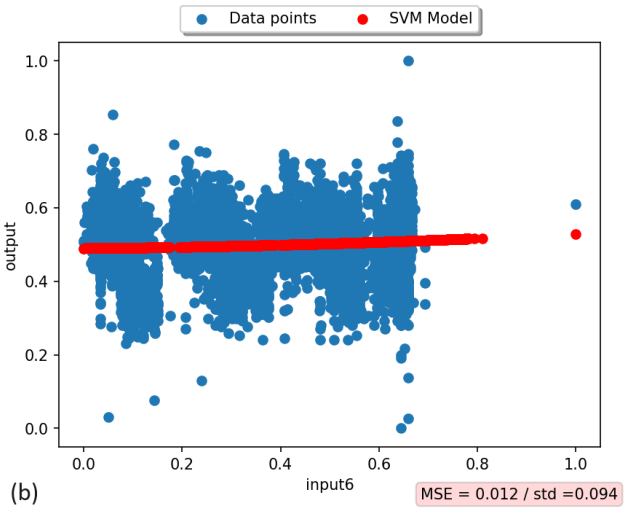
4 Conclusion

To identify the most effective model for precise steel quality prediction, we conduct a brief overview of our progress thus far. Among various linear regression models explored, the main model (without changes in slope and intercept) exhibited the most favorable fit. Second-degree polynomial linear regression method resulted in the lowest MSE among others. The SVM model achieved the minimum MSE when all hyperparameters were set to 0.1.

Table 3: *Error metrics of polynomial regression models for input6 vs. the output.*

Plot	Degree	MSE	RMSE	STD
a	2	0.014916	0.122131	0.094451
b	3	0.014765	0.121511	0.09433
c	4	0.017174	0.13105	0.097009





C	gamma	epsilon	mean_test_score
0.1	0.1	0.1	-0.0035352

C	gamma	epsilon	mean_test_score
0.1	0.1	0.1	-0.00713088

Figure 11: *SVM models of (a) input12 against input21, and (b) input6 against the output.*

Upon comparing the MSE values for input12 vs. input21 variables, it is found that the second-degree polynomial linear regression model has achieved the lowest MSE among the models investigated. On the other hand, considering variables input6 and the output, SVM method has yielded the lowest MSE among the models investigated.

In general, based on the results shown in this report and also the results obtained from applying the same ML algorithms to other variables within the dataset, it is concluded that second-degree polynomial regression and SVM model are the most effective methods for accurately predicting steel quality features.

Throughout this project different sources have been used, such as the book Raschka, 2015, information from websites, and insights from ChatGPT. All these references are listed in the References section.

4.1 Answers to research questions

With developing ML models:

- The accuracy and efficiency of steel production can be improved by predicting steel quality in real-time. With analyzing historical data and various parameters related to steel production processes, ML models can forecast steel quality metrics, which helps make changes before problems occure and this leads to improve quality during production.
- Material wastage and economic losses can be minimized by preemptively identifying potential quality deviations through accurate predictions. With analyzing historical data and various factors that contribute to material quality, ML models can forecast potential deviations or defects in the manufacturing process. Moreover, by understanding the relationships between process parameters and material quality, ML models can recommend optimal conditions to maintain con-

sistent quality, and consequently, reduce the likelihood of deviations.

- State-of-the-art methods can be integrated and benchmarks can be set for predictive accuracy and efficiency. By utilizing cutting-edge machine learning algorithms predictive accuracy is improved. In addition, model performance can be optimized by fine-tuning hyperparameters using techniques like grid search or random search to identify the best combination of parameters for improved accuracy. At the end, the model's performance against existing benchmarks and industry standards can be compared to demonstrate its superiority or to establish new benchmarks in predictive accuracy and efficiency.

References

Dealing with outliers using the Z-Score method (n.d.). https://www.analyticsvidhya.com/blog/2022/08/dealing-with-outliers-using-the-z-score-method/.

how to check the .csv dataset for any missing valuse, and then to drop rows/columns with missing values— fill missing values using mean and median— how to find non-numeric data from a .csv file, and then to convert it to numerical— tell me about z_scores and how to use it— consider an excel file (.csv format)use DBSCAN method to find outliers, and then replace them with mean— what is correlation_matrix?— create hystograms using subplot— how to create normal probability plots?— write a Python code for linear regression— write me a multiple linear regression with importing data from a .csv file using scikit-learn— plot all inputs and regression line— imagine thet we have to datasets one for training (.csv format) the other for testing (.csv format). use a polynomial regression model to predict, and then plot them.— if I have 25 columns of inputs with one column of output (.csv file) give me 25 plots in a 5 by 5 area— create a svm model with Python— How to create a gid search (n.d.). https://chat.openai.com/.

How to Read and Write to CSV Files in Python (n.d.). https://medium.com/codex/how-to-read-and-write-to-csv-files-in-python-380dabec30b4.

How to Read and Write With CSV Files in Python? (N.d.). https://www.analyticsvidhya.com/blog/2021/08/python-tutorial-working-with-csv-file-for-data-science/.

Linear Regression in Python (n.d.). https://realpython.com/linear-regression-in-python/.

Linear Regression (Python Implementation) (n.d.). https://www.geeksforgeeks.org/linear-regression-python-implementation/.

Outliers Detection Using IQR, Z-score, LOF and DBSCAN (n.d.). https://www.analyticsvidhya.com/blog/2022/10/outliers-detection-using-iqr-z-score-lof-and-dbscan/.

Pairs plot (pairwise plot) in seaborn with the pairplot function (n.d.). https://python-charts.com/correlation/pairs-plot-seaborn/?utm_content=cmp-true.

Raschka, Sebastian (2015). *Python machine learning*. Packt publishing ltd.

Z score for Outlier Detection – Python (n.d.). https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/.