DS203 : Exercise 3

Nirav Bhattad

Last updated September 1, 2024

Step 1

Step 1

Review the Jupyter Notebook `E3.ipynb` and:
(a) Create a summary of the code therein.
(b) Are there any learnings from this code that you wish to highlight?

This Python Notebook `E3.ipynb` presents an analysis of various regression models applied to a dataset using polynomial features of different degrees. The models analyzed include Linear Regression, Support Vector Machine (SVM) Regression, Random Forest, Gradient Boosting, K-Nearest Neighbors (KNN), and Neural Networks. The performance of each model is evaluated based on metrics such as R-squared, Mean Squared Error (MSE), Durbin-Watson, and Jarque-Bera statistics.

We use the dataset given in `E3-MLR3.xlsx`. The dataset contains 548 samples of (y, x_1) to train the model and 152 samples of (y, x_1) to test the model.

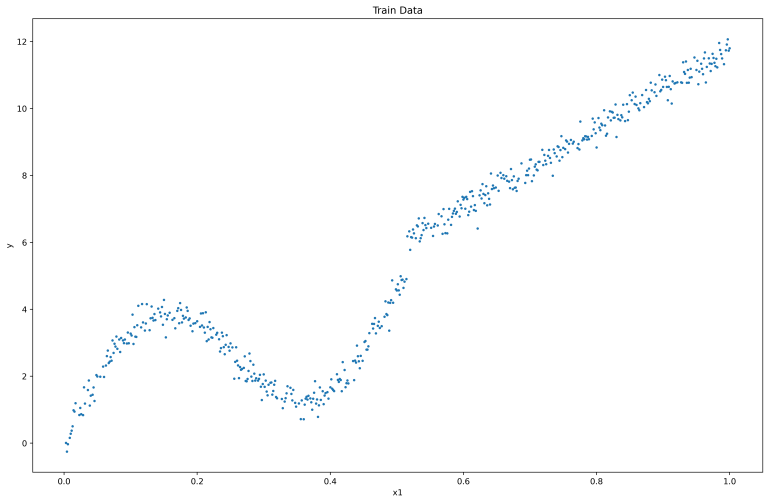

Figure 1: Training Data used in the analysis

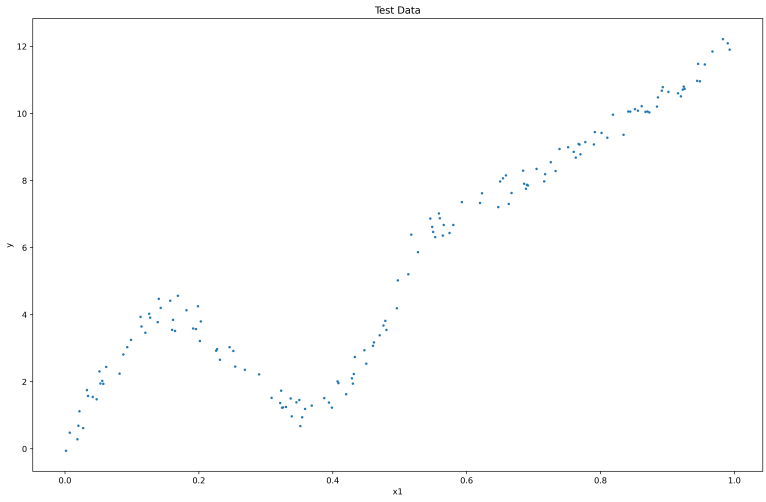

Figure 2: Testing Data used in the analysis

The following Python script in `E3.ipynb` applies multiple regression algorithms to a dataset, evaluates their performance, and visualizes the results.

1. Importing Necessary Libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

Listing 1: Importing Libraries

Description: The code begins by importing essential libraries for data manipulation (`pandas`, `numpy`), machine learning models (`scikit-learn`), statistical analysis (`statsmodels`), and plotting (`matplotlib`).

2. Data Loading and Preprocessing

```
file_path = 'E3-MLR3.xlsx'
train_data = pd.read_excel(file_path, sheet_name='train')
test_data = pd.read_excel(file_path, sheet_name='test')

X_train = train_data.drop(columns=['y'])
y_train = train_data['y']

X_test = test_data.drop(columns=['y'])
y_test = test_data['y']

poly = PolynomialFeatures(degree=6)
X_train_poly = poly.fit_transform(X_train)
```

```
X_test_poly = poly.transform(X_test)
```

Listing 2: Loading and Preprocessing Data

Description: The script reads training and testing data from an Excel file and separates features (`X_train`, `X_test`) and the target variable (`y_train`, `y_test`). It then uses `PolynomialFeatures` to augment the features with polynomial terms of degree 6.

3. Saving the Augmented Dataset

```
feature_names = poly.get_feature_names_out(X_train.columns)
augmented_data = pd.DataFrame(X_test_poly, columns=feature_names)
augmented_data['y'] = test_data['y']
augmented_data.to_csv('augmented_data.csv', index=False)
```

Listing 3: Saving Augmented Data

Description: The code generates a CSV file containing the test data with the newly created polynomial features for review.

4. Defining and Running Regression Models

```
algorithms = {
    'Linear Regression': LinearRegression(),
    'SVM Regression': SVR(kernel='poly'),
    'RandomForest': RandomForestRegressor(),
    'XGBoost': GradientBoostingRegressor(),
    'knn': KNeighborsRegressor(),
    'Neural Network': MLPRegressor(hidden_layer_sizes=[10,10,10], max_iter=20000)
}

metric_table_train = pd.DataFrame()
metric_table_test = pd.DataFrame()

fig, axs = plt.subplots(len(algorithms), 4, figsize=(20, 20))
fig_row = -1

for algorithm_name, algorithm in algorithms.items():
    algorithm.fit(X_train_poly, y_train)
    y_train_pred = algorithm.predict(X_train_poly)
    y_test_pred = algorithm.predict(X_test_poly)

    r2_train = algorithm.score(X_train_poly, y_train)
    mse_train = mean_squared_error(y_train, y_train_pred)
    r2_test = algorithm.score(X_test_poly, y_test)
    mse_test = mean_squared_error(y_test, y_test_pred)

    residuals_train = y_train - y_train_pred
    residuals_test = y_test - y_test_pred

    durbin_watson_stat_train = sm.stats.durbin_watson(residuals_train)
    jb_stat_train, jb_p_value_train, _, _ = sm.stats.jarque_bera(residuals_train)

    durbin_watson_stat_test = sm.stats.durbin_watson(residuals_test)
    jb_stat_test, jb_p_value_test, _, _ = sm.stats.jarque_bera(residuals_test)

    metric_table_train.at[algorithm_name, 'R-squared'] = r2_train
    metric_table_train.at[algorithm_name, 'MSE'] = mse_train
    metric_table_train.at[algorithm_name, 'Durbin-Watson'] = durbin_watson_stat_train
    metric_table_train.at[algorithm_name, 'Jarque-Bera'] = jb_stat_train
    metric_table_train.at[algorithm_name, 'JB P-value'] = jb_p_value_train

    metric_table_test.at[algorithm_name, 'R-squared'] = r2_test
    metric_table_test.at[algorithm_name, 'MSE'] = mse_test
    metric_table_test.at[algorithm_name, 'Durbin-Watson'] = durbin_watson_stat_test
```

```
    metric_table_test.at[algorithm_name, 'Jarque-Bera'] = jb_stat_test
    metric_table_test.at[algorithm_name, 'JB P-value'] = jb_p_value_test

    fig_row = fig_row+1

    axs[fig_row, 0].scatter(train_data['x1'], y_train)
    axs[fig_row, 0].scatter(train_data['x1'], y_train_pred)
    axs[fig_row, 0].set_title(algorithm_name + " - Train")

    axs[fig_row, 1].scatter(train_data['x1'], residuals_train)
    axs[fig_row, 1].set_title(algorithm_name + " Residuals - Train")

    axs[fig_row, 2].scatter(test_data['x1'], y_test)
    axs[fig_row, 2].scatter(test_data['x1'], y_test_pred)
    axs[fig_row, 2].set_title(algorithm_name + " - Test")

    axs[fig_row, 3].scatter(test_data['x1'], residuals_test)
    axs[fig_row, 3].set_title(algorithm_name + " Residuals - Test")
```

Listing 4: Defining and Running Regression Models

Description: A dictionary of regression algorithms is created. Each algorithm is trained on the polynomial features, and its performance is evaluated using metrics such as R-squared, Mean Squared Error (MSE), Durbin-Watson, and Jarque-Bera test. These metrics are stored in tables. The residuals and predicted values are also plotted.

5. Visualizing the Results

```
plt.tight_layout()
plt.show()
```

Listing 5: Visualizing the Results

Description: The code concludes with displaying the generated plots for each regression model.

Learnings from this Code

(a) The code demonstrates the application of various regression algorithms to a dataset with polynomial features.

(b) It highlights the importance of evaluating regression models using multiple metrics to understand their performance.

(c) The code showcases the use of residual analysis to assess the model's assumptions and identify potential issues.

(d) It provides a structured approach to comparing different regression models and their outcomes.

(e) The code emphasizes the significance of visualizing the model's predictions and residuals for better interpretation.

(f) The code also shows how to save augmented datasets for further analysis or review.

Step 2

Step 2

Review the **Sklearn** documentation for each Sklearn function used in the Notebook (eg. `PolynomialFeatures`, `LinearRegression`, `mean_squared_error`, etc.) and create a description of each to explain, to yourself, the functionality, the input parameters, and the outputs generated. Present this in the form of a two-column - Table (Function name | Description).

`PolynomialFeatures`	This function is used to generate polynomial and interaction features. It generates a new feature matrix consisting of all polynomial combinations of the features with a degree less than or equal to the specified degree. The input to this function is the degree of the polynomial features to be generated. The output generated is a new feature matrix consisting of all polynomial combinations of the features with a degree less than or equal to the specified degree. Input Parameters: • **degree**: `int` or `tuple (min_degree, max_degree)`, `default=2` • **interaction_only**: `bool`, `default=False` • **include_bias**: `bool`, `default=True` • **order**: `str` in {'C', 'F'}, `default='C'` Attributes: • **powers_**: `ndarray` of shape (`n_output_features_`, `n_input_features_`) • **n_output_features_**: `int` • **n_features_in_**: `int` • **feature_names_in_**: `ndarray` of shape (`n_input_features_`,) Output: • **ndarray** of shape (`n_samples`, `n_output_features_`)
`LinearRegression`	This function is used to fit a linear model. It fits a linear model with coefficients $w = (w_1, ..., w_p)$ to minimize the residual sum of squares between the observed targets in the dataset and the targets predicted by the linear approximation. Input Parameters: • **fit_intercept**: `bool`, `default=True` • **copy_X**: `bool`, `default=True` • **n_jobs**: `int`, `default=None` • **positive**: `bool`, `default=False` Attributes: • **coef_**: `ndarray` of shape (`n_targets`, `n_features`) • **intercept_**: `ndarray` of shape (`n_targets`,) • **rank_**: `int` • **singular_**: `ndarray` of shape (`min(X, y)`,) • **n_features_in_**: `ndarray` of shape (`n_targets`,) Output: • **self**: returns an instance of self

`SVR`	This function is used to fit the Support Vector Regression model. Input Parameters: • **kernel**: `str`, `default='rbf'` • **degree**: `int`, `default=3` • **gamma**: `float`, `default='scale'` • **coef0**: `float`, `default=0.0` • **tol**: `float`, `default=1e-3` • **C**: `float`, `default=1.0` • **epsilon**: `float`, `default=0.1` • **shrinking**: `bool`, `default=True` • **cache_size**: `float`, `default=200` • **verbose**: `bool`, `default=False` • **max_iter**: `int`, `default=-1` Attributes: • **coef_**: `ndarray` of shape (1, `n_features`) • **dual_coef_**: `ndarray` of shape (1, `n_SV`) • **fit_status_**: `int` • **intercept_**: `ndarray` of shape (1,) • **n_features_in_**: `int` • **feature_names_in_**: `ndarray` of shape (`n_features`,) • **n_iter_**: `int` • **n_support_**: `ndarray` of shape (1,) • **shape_fit_**: `tuple` of `int` of shape (`n_dimensions_of_X`,) • **support_**: `ndarray` of shape (`n_SV`,) • **support_vectors_**: `ndarray` of shape (`n_SV`, `n_features`) Output: • **ndarray** of shape (`n_samples`,)
`KNeighborsRegressor`	This function is used to fit the K-Nearest Neighbors regressor model, which is a non-parametric method used for regression. It predicts the target variable by learning from the k nearest data points in the feature space. Input Parameters: • **n_neighbors**: `int`, `default=5` • **weights**: `str`, `default='uniform'` • **algorithm**: `str`, `default='auto'` • **leaf_size**: `int`, `default=30` • **p**: `int`, `default=2` • **metric**: `str`, `default='minkowski'` • **metric_params**: `dict`, `default=None` • **n_jobs**: `int`, `default=None` Attributes: • **effective_metric_**: `str` • **effective_metric_params_**: `dict` • **n_samples_fit_**: `int` • **n_features_in_**: `int` • **feature_names_in_**: `ndarray` of shape (`n_features`,) Output: • **ndarray** of shape (`n_samples`,)

`RandomForestRegressor`	This function is used to fit a random forest model, which is an ensemble learning method for regression. It fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Input Parameters: • **n_estimators**: `int`, `default=100` • **criterion**: `str`, `default='squared_error'` • **max_depth**: `int`, `default=None` • **min_samples_split**: `int`, `default=2` • **min_samples_leaf**: `int`, `default=1` • **min_weight_fraction_leaf**: `float`, `default=0.0` • **max_features**: `int`, `default='auto'` • **max_leaf_nodes**: `int`, `default=None` • **min_impurity_decrease**: `float`, `default=0.0` • **bootstrap**: `bool`, `default=True` • **oob_score**: `bool`, `default=False` • **n_jobs**: `int`, `default=None` • **random_state**: `int`, `default=None` • **verbose**: `int`, `default=0` • **warm_start**: `bool`, `default=False` • **ccp_alpha**: `float`, `default=0.0` • **max_samples**: `int`, `default=None` • **monotonic_cst**: `array-like` of shape (`n_features`), `default=None` Attributes: • **estimator_**: `DecisionTreeRegressor` • **estimators_**: `list of DecisionTreeRegressor` • **n_features_in**: `int` • **feature_names_in_**: `ndarray` of shape (`n_features`,) • **n_outputs_**: `int` • **oob_score_**: `ndarray` of shape (`n_estimators`,) • **oob_prediction_**: `ndarray` of shape (`n_samples`, `n_outputs`) • **estimators_samples_**: `list of ndarray` Output: • **ndarray** of shape (`n_samples`,)
`mean_squared_error`	This function is used to compute the mean squared error regression loss. It computes the mean squared error between the true and predicted values. Input Parameters: • **y_true**: `ndarray` of shape (`n_samples`,) • **y_pred**: `ndarray` of shape (`n_samples`,) • **squared**: `bool`, `default=True` • **multioutput**: `str`, `default='uniform_average'` • **sample_weight**: `ndarray` of shape (`n_samples`,), `default=None` Output: • **float**

GradientBoostingRegressor	This function is used to fit a gradient boosting model, which is an ensemble learning method for regression. It fits a number of regression trees on various sub-samples of the dataset and uses boosting to improve the predictive accuracy and control over-fitting. Input Parameters: • **loss**: `str`, `default='ls'` • **learning_rate**: `float`, `default=0.1` • **n_estimators**: `int`, `default=100` • **subsample**: `float`, `default=1.0` • **criterion**: `str`, `default='friedman_mse'` • **min_samples_split**: `int`, `default=2` • **min_samples_leaf**: `int`, `default=1` • **min_weight_fraction_leaf**: `float`, `default=0.0` • **max_depth**: `int`, `default=3` • **min_impurity_decrease**: `float`, `default=0.0` • **init**: `estimator`, `default=None` • **random_state**: `int`, `default=None` • **max_features**: `str`, `default=None` • **alpha**: `float`, `default=0.9` • **verbose**: `int`, `default=0` • **max_leaf_nodes**: `int`, `default=None` • **warm_start**: `bool`, `default=False` • **validation_fraction**: `float`, `default=0.1` • **n_iter_no_change**: `int`, `default=None` • **tol**: `float`, `default=1e-4` • **ccp_alpha**: `float`, `default=0.0` Attributes: • **n_estimators_**: `int` • **n_trees_per_iteration_**: `int` • **feature_importances_**: `ndarray` of shape (`n_features`,) • **oob_improvement_**: `ndarray` of shape (`n_estimators`,) • **oob_score_**: `ndarray` of shape (`n_estimators`,) • **oob_scores_**: `ndarray` of shape (`n_estimators`,) • **train_score_**: `ndarray` of shape (`n_estimators`,) • **init_**: `estimator` • **estimators_**: `ndarray of DecisionTreeRegressor` • **n_features_in_**: `int` • **feature_names_in_**: `ndarray` of shape (`n_features`,) • **max_features_**: `int` Output: • **ndarray** of shape (`n_samples`,)

`r2_score`	This function is used to compute the R-squared regression score function. It computes the R-squared score, which is the coefficient of determination. This function has not been used in the jupyter notebook. Input Parameters: • **y_true**: `ndarray` of shape (`n_samples`,) • **y_pred**: `ndarray` of shape (`n_samples`,) • **sample_weight**: `ndarray` of shape (`n_samples`,), `default=None` • **multioutput**: `str`, `default='uniform_average'` • **force_finite**: `bool`, `default=True` Output: • **float**

`MLPRegressor`	This function is used to fit a Multi-layer Perceptron regressor. It trains a neural network model with a single hidden layer, which is a fully-connected feed-forward artificial neural network. Input Parameters: • **hidden_layer_sizes**: `tuple`, `default=(100,)` • **activation**: `str`, `default='relu'` • **solver**: `str`, `default='adam'` • **alpha**: `float`, `default=0.0001` • **batch_size**: `int`, `default='auto'` • **learning_rate**: `str`, `default='constant'` • **learning_rate_init**: `float`, `default=0.001` • **power_t**: `float`, `default=0.5` • **max_iter**: `int`, `default=200` • **shuffle**: `bool`, `default=True` • **random_state**: `int`, `default=None` • **tol**: `float`, `default=1e-4` • **verbose**: `bool`, `default=False` • **warm_start**: `bool`, `default=False` • **momentum**: `float`, `default=0.9` • **nesterovs_momentum**: `bool`, `default=True` • **early_stopping**: `bool`, `default=False` • **validation_fraction**: `float`, `default=0.1` • **beta_1**: `float`, `default=0.9` • **beta_2**: `float`, `default=0.999` • **epsilon**: `float`, `default=1e-8` • **n_iter_no_change**: `int`, `default=10` • **max_fun**: `int`, `default=15000` Attributes: • **loss_**: `float` • **best_loss_**: `float` • **loss_curve_**: `list` of shape (`n_iter_`,) • **t_**: `int` • **validation_scores_**: `list` of shape (`n_iter_`,) • **best_validation_score_**: `float` • **coefs_**: `list` of shape (`n_layers_ - 1`,) • **intercepts_**: `list` of shape (`n_layers_ - 1`,) • **n_features_in_**: `int` • **feature_names_in_**: `ndarray` of shape (`n_features`,) • **n_iter_**: `int` • **n_layers_**: `int` • **n_outputs_**: `int` • **out_activation_**: `str` Output: • **ndarray** of shape (`n_samples`,)

Step 3

Step 3

Generate outputs by setting **degree = 1**, **degree = 3**, **degree = 6**, **degree = 10**, in the `PolynomialFeatures` function used in `E3.ipynb` and analyze them as follows:

(a) Review the `augmented_data.csv` file generated in each case and document your observations.
(b) Create an overall qualitative summary based on a review and analysis of the Figures generated.
(c) Summarize and explain the variations in the metrics **across regression methods for a given degree** (ie. a given set of polynomial features). Cover both, train and test, metrics, and compare them.
(d) Summarize and explain the variations in the metrics **across degrees for a given regression method**. Cover both, train, and test metrics, and compare them.
(e) When **degree = 1** which method(s) result in acceptable regression models? Why?
(f) When **degree = 6** which method(s) result in acceptable regression models? Why?
(g) As the value of degree is increased to 10 which regression methods show the most impact? Why?
(h) Why do non-parametric methods like `KNN` and `Decision Tree` based methods generate good results even without feature engineering?
(i) What are the limitations of the non-parametric methods?
(j) Given the results, should LinearRegression be used at all? Why, when? Justify your answer.

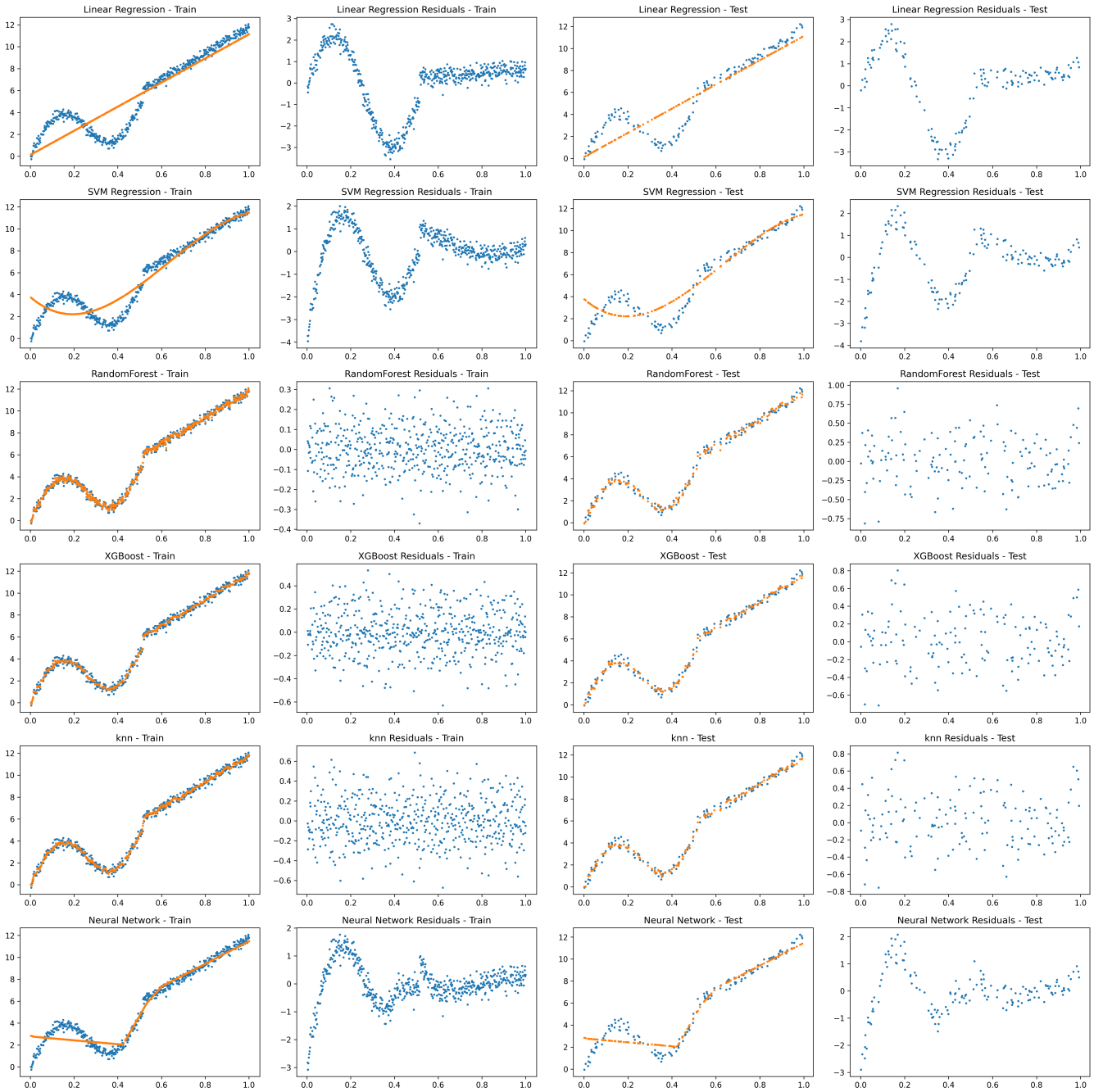

Figure 3: Various Regression Models with degree = 1

When degree = 1, the following observations can be made:

•

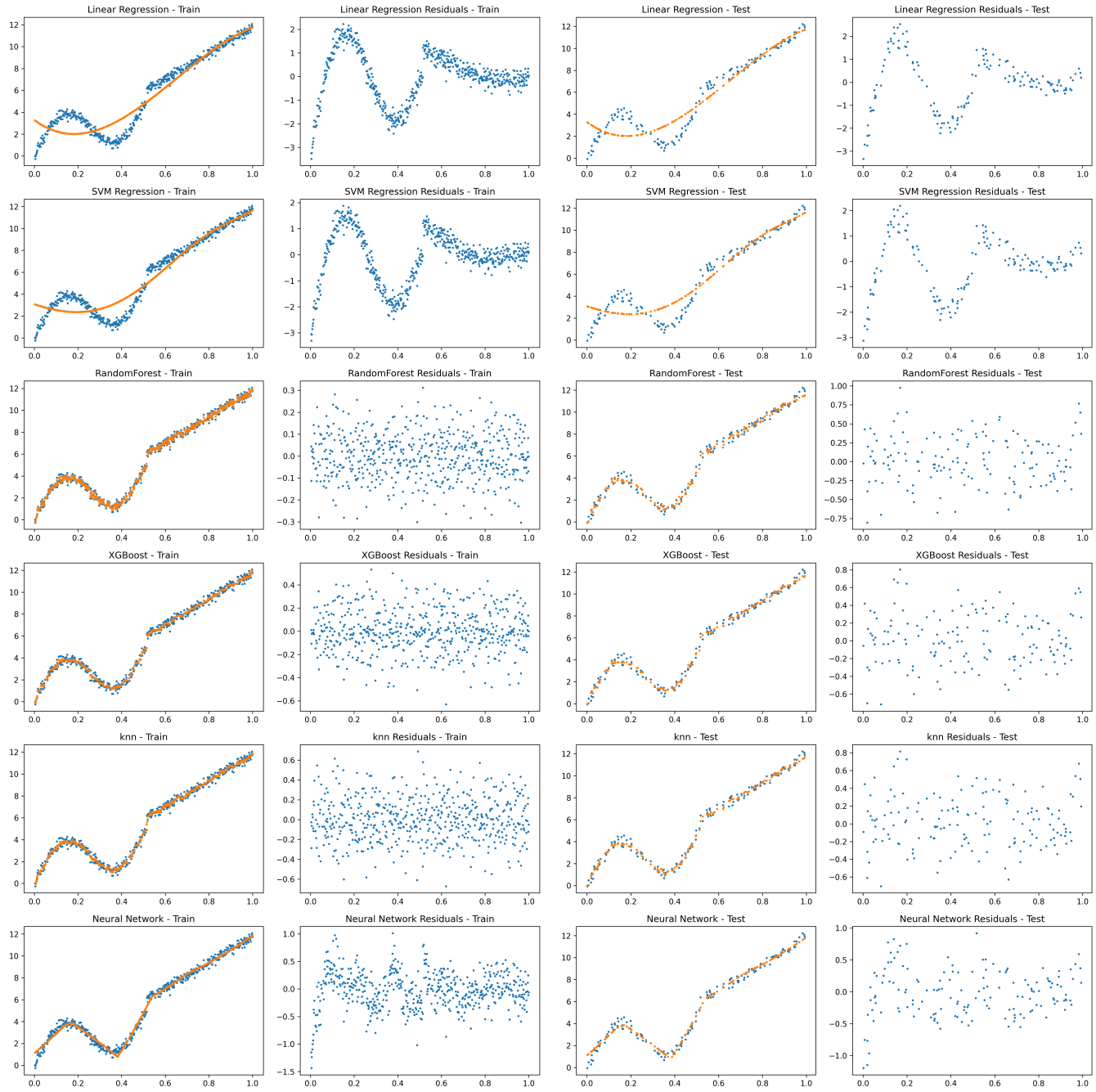

Figure 4: Various Regression Models with degree = 3

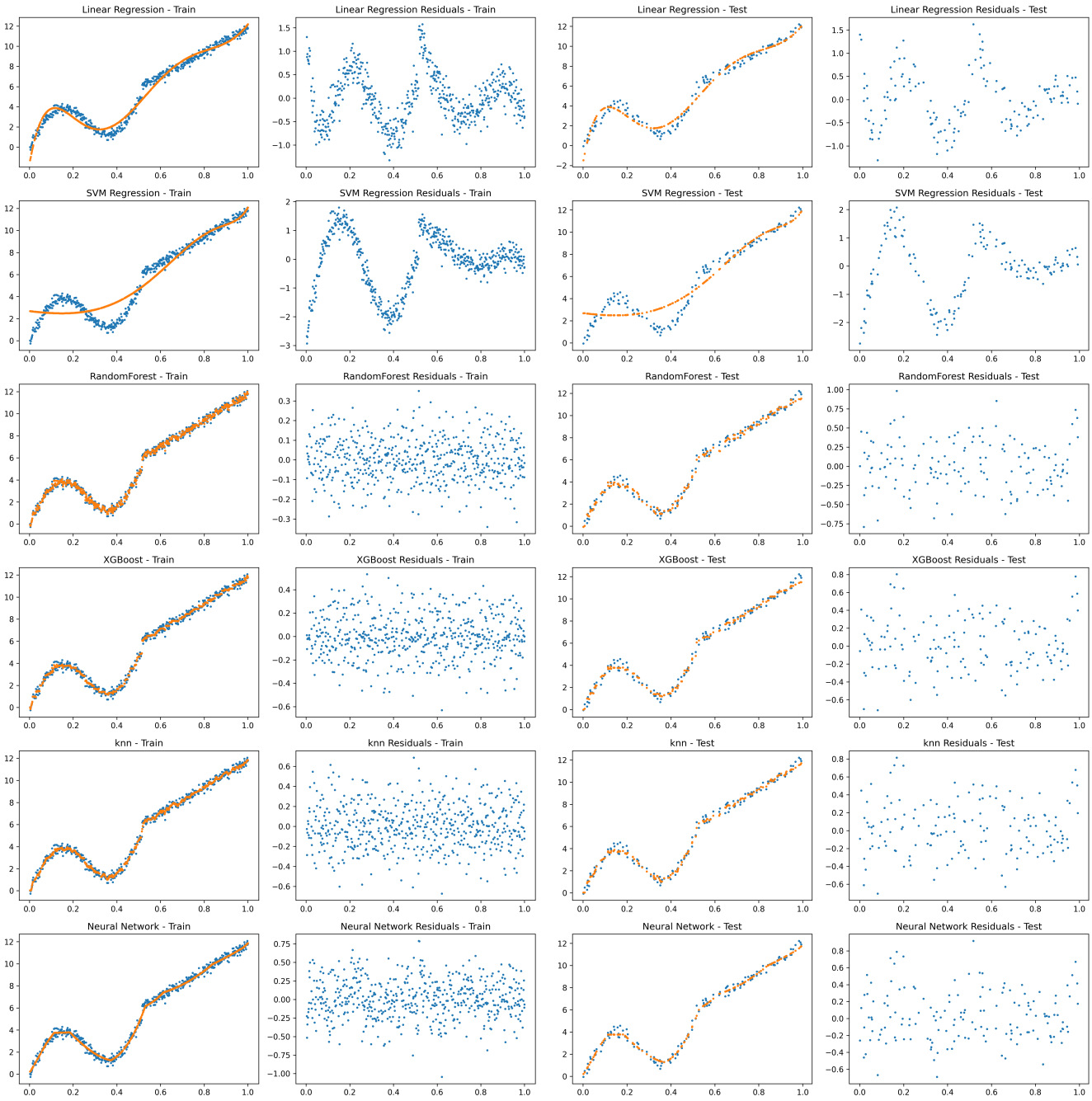

Figure 5: Various Regression Models with degree = 6

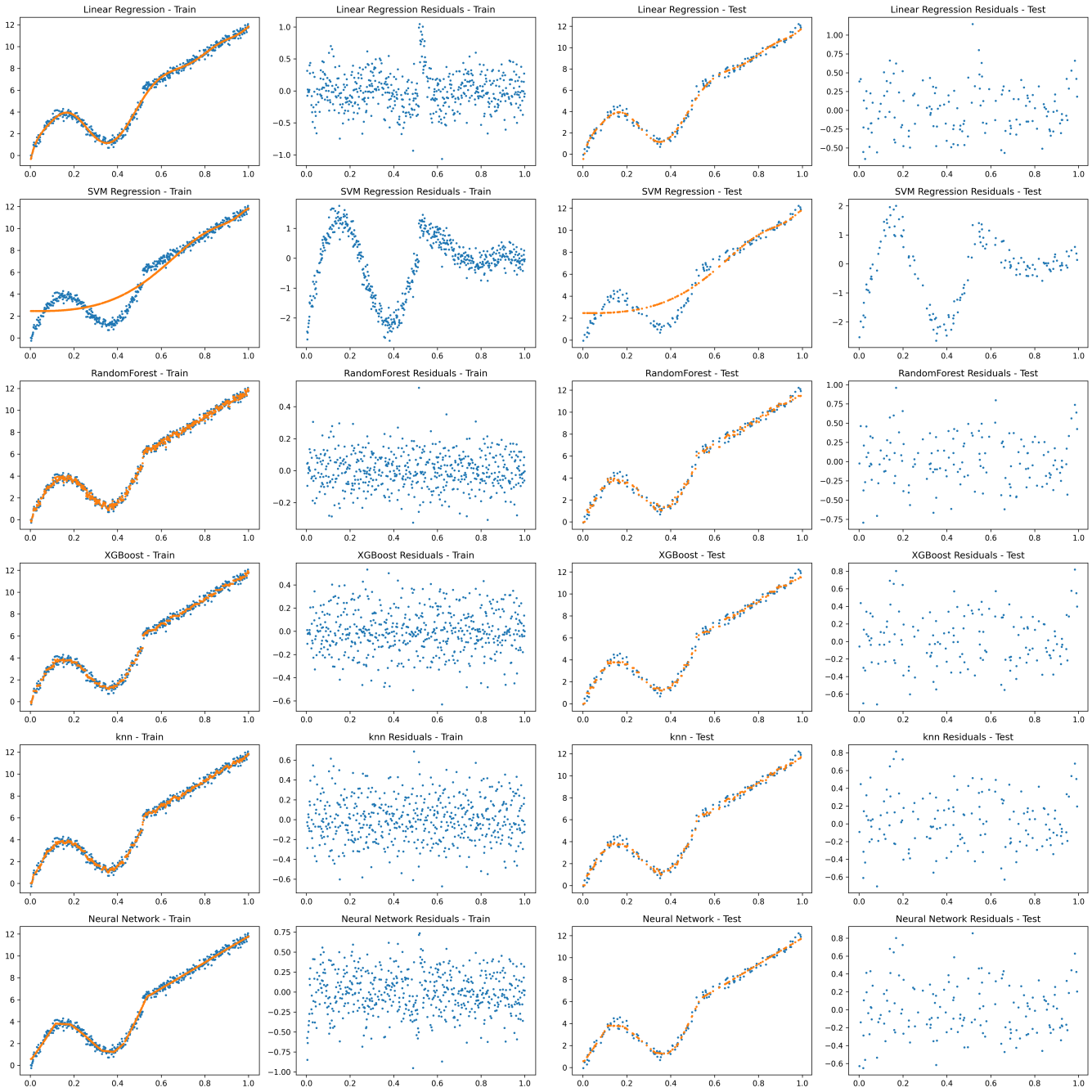

Figure 6: Various Regression Models with degree = 10

Question (a)

Review the `augmented_data.csv` file generated in each case and document your observations.

Question (b)

Create an overall qualitative summary based on a review and analysis of the Figures generated.

Question (c)

Summarize and explain the variations in the metrics **across regression methods for a given degree** (ie. a given set of polynomial features). Cover both, train and test, metrics, and compare them.

Question (d)

Summarize and explain the variations in the metrics **across degrees for a given regression method**. Cover both, train, and test metrics, and compare them.

Question (e)

When **degree = 1** which method(s) result in acceptable regression models? Why?

Question (f)

When **degree = 6** which method(s) result in acceptable regression models? Why?

Question (g)

As the value of degree is increased to 10 which regression methods show the most impact? Why?

Question (h)

Why do non-parametric methods like `KNN` and `Decision Tree` based methods generate good results even without feature engineering?

Question (i)

What are the limitations of the non-parametric methods?

Question (j)

Given the results, should LinearRegression be used at all? Why, when? Justify your answer.

Step 4

Step 4

In step 2 you have already reviewed the important parameters and outputs related to the regression methods. Select 2-3 methods, vary the important parameters, and observe how the outputs change (eg. see the function calls for `SVR` and `MLPRegressor`). Document the outcomes of your experiments.

Step 5

Step 5

Review **`Sklearn`** documentation to understand and experiment with a few more (2-3) regression methods, in addition to the ones listed above, and document the outcomes of your experiments.

Main Learnings