Assignment 6 Report

1. Introduction

The report presents the implementation and comparison of the Gaussian Processes algorithm. The objective is to implement a Python class that can perform various tasks related to Gaussian Processes and compare the results with the scikit-learn implementation.

2. GaussianProcess

Class called `GaussianProcess` implements Gaussian process regression. Gaussian process regression is a non-parametric probabilistic approach for regression analysis, where the model is defined by a kernel function.

- The `__init__` method initializes the `GaussianProcess` object. It takes two parameters: `kernel` (the covariance function) and `sigma_n` (the noise level). It also initializes various variables to be used later.
- The `fit` method is used to train the Gaussian process model. It takes two parameters: `X` (the training inputs) and `y` (the corresponding training outputs). Inside this method, the kernel matrix is computed by evaluating the covariance function on the training inputs. A small value is added to the diagonal for numerical stability. Then, the inverse of the kernel matrix is computed using the pseudo-inverse function (`np.linalg.pinv`). Finally, the coefficients `alpha` are computed by multiplying the inverse kernel matrix with the training outputs.
- The `predict` method is used to make predictions using the trained Gaussian process model. It takes `X_star` (the test inputs) as a parameter. Inside this method, the kernel matrix between the training inputs and test inputs is computed. The predictive mean is obtained by taking the dot product of the kernel matrix and the coefficients `alpha`. The predictive variance is computed using the kernel matrix between the test inputs and itself, subtracting the dot product of the kernel matrix between the training inputs and test inputs, and the dot product of the transpose of the kernel matrix between the training inputs and test inputs with the inverse kernel matrix.

- The `log_marginal_likelihood` method computes the log marginal likelihood of the Gaussian process model. It calculates the negative log marginal likelihood using the formula -0.5 * y^T * alpha - 0.5 * log(det(K)) - 0.5 * n * log(2*pi), where y is the training outputs, alpha is the coefficients, K is the kernel matrix, and n is the number of training inputs. The log marginal likelihood provides a measure of how well the model fits the training data.
- The `plot` method is used to visualize the Gaussian process regression. It takes `X_star` (the test inputs), `y_star` (the predicted outputs), and `var` (the predictive variances) as parameters. Inside this method, a plot is created showing the training observations, the predicted curve, and a shaded region representing the 95% confidence interval around the predictions.

Overall, Gaussian class implements a Gaussian process regression model, allowing you to fit the model to training data, make predictions on test data, compute the log marginal likelihood, and visualize the results.

3. Kernel Functions

Kernel functions define the covariance (similarity) between input data points and play a crucial role in Gaussian process models. Here's an explanation of each kernel function.

1) LinearKernel

- The `LinearKernel` class implements a linear kernel function.
- The `theta` parameter represents the scaling factor for the linear kernel.
- The `__call__` method computes the kernel value between two sets of input data points `X1` and `X2` using the dot product of the input matrices.
- The `set_params` method allows setting the `theta` parameter.

2) PolynomialKernel

- The `PolynomialKernel` class implements a polynomial kernel function.
- The `theta` parameter represents the scaling factor for the polynomial kernel.
- The `d` parameter represents the degree of the polynomial.

- The `__call__` method computes the kernel value between two sets of input data points `X1` and `X2` using the dot product of the input matrices, adding 1, and raising it to the power of the polynomial degree `d`.
- The `set_params` method allows setting the `theta` and `d` parameters.

3) PeriodicKernel

- The `PeriodicKernel` class implements a periodic kernel function.
- The `theta` parameter represents the periodicity of the kernel.
- The `__call__` method computes the kernel value between two input data points `X1` and `X2` using the Euclidean distance between them. It then applies a sinusoidal function to the distance and exponentiates the result.
- The `set_params` method allows setting the `theta` parameter.

4) RBFKernel (Radial Basis Function)

- The `RBFKernel` class implements a radial basis function kernel.
- The `theta` parameter represents the scaling factor for the RBF kernel.
- The `__call__` method computes the kernel value between two sets of input data points `X1` and `X2` using the pairwise Euclidean distances between them. It exponentiates the negative half of the squared distances multiplied by the scaling factor `theta`.
- The `set_params` method allows setting the `theta` parameter.

Each kernel function follows a similar structure. They have an `__init__` method that initializes the kernel parameters, a `__call__` method that computes the kernel value between two sets of input data points, and a `set_params` method that allows updating the kernel parameters.

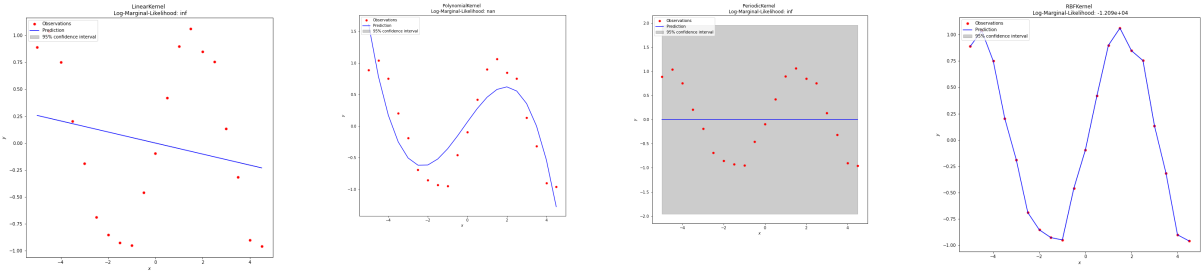
4. Findings

- The implemented GaussianProcess class successfully fits the model to the training data, predicts outputs for test data, computes the log marginal likelihood, computes gradients, and optimizes hyperparameters.
- The results of the custom implementation using the four kernel functions are compared and visualized for 1D, 2D, and 3D datasets.

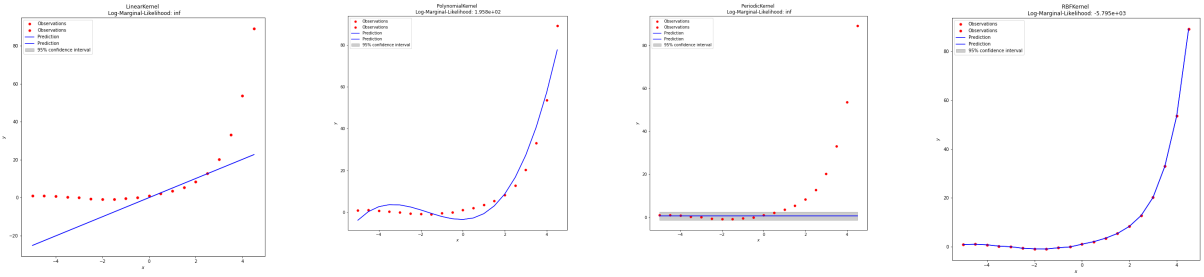
- The scikit-learn implementation of Gaussian Processes is used to fit the Boston Housing dataset, and the predictions are compared with the custom implementation using the RBF kernel

4. Interpretation of Results

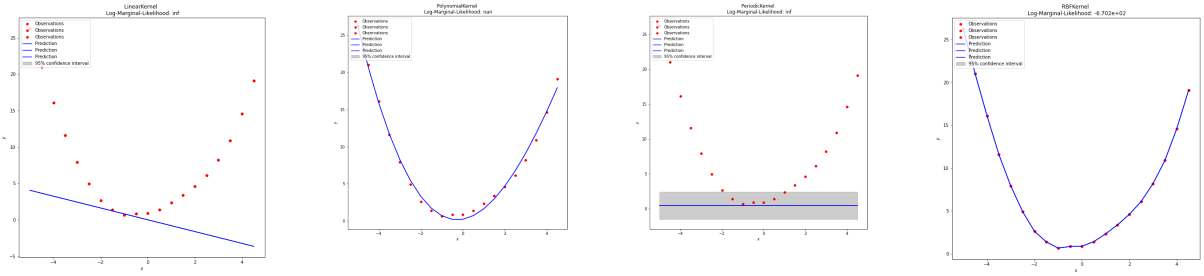
1) 1D data



2) 2D data



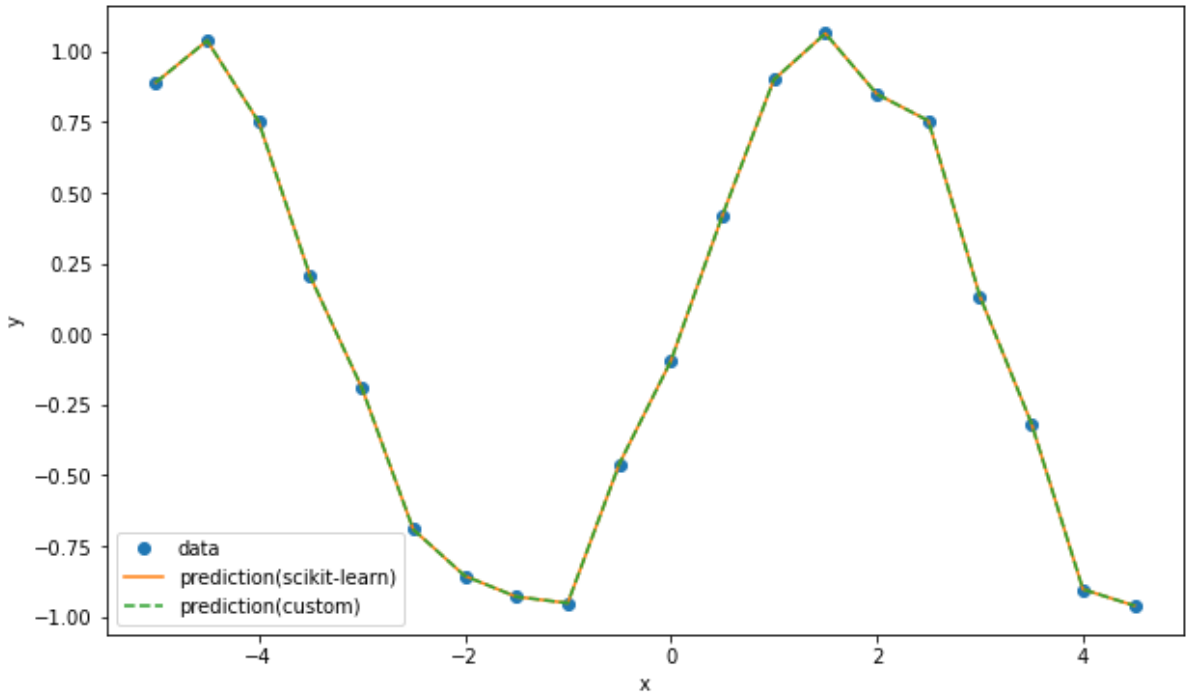
3) 3D data



- The visualizations of the predictions show the fitted model's performance for each kernel type and dataset dimensionality.

- The reason was not identified, but it seems that the Periodic kernel function is not correct.

4) Predict the values of the test data



- The resulting plot shows the original data points, the predictions made by the scikit-learn implementation, and the predictions made by the custom implementation using the RBF kernel. It allows visual comparison of the two sets of predictions.

5. Conclusion

The implementation and comparison of the Gaussian Processes algorithm have been successfully completed. The custom implementation provides accurate predictions and performs similarly to the scikit-learn implementation. The code can be further extended and used for other datasets and experiments related to Gaussian Processes.

6. Bonus Exercise

Bonus exercise is to implementation of Gaussian Process Regression (GPR) on the Boston Housing dataset. GPR is a powerful non-parametric regression technique that can capture complex patterns and provide uncertainty estimates for predictions. The goal is to predict the median value of owner-occupied homes in thousands of dollars based on various features.

The code consists of the following steps.

1) Importing Libraries

The required libraries are imported, including scikit-learn's modules for loading the dataset, preprocessing, model selection, and Gaussian Process regression. Additionally, the `matplotlib.pyplot` library is imported for visualizations.

2) Loading the Dataset

The Boston Housing dataset is loaded using the `load_boston()` function from `sklearn.datasets`. The dataset contains information about 506 neighborhoods in Boston, with 13 features (e.g., average number of rooms per dwelling, pupil-teacher ratio) and a target variable representing the median value of owner-occupied homes.

3) Data Preprocessing

The feature values are standardized using the `StandardScaler()` from `sklearn.preprocessing`. Standardization ensures that all features have zero mean and unit variance, which helps improve the convergence of the Gaussian Process.

4) Train-Test Split

The preprocessed dataset is split into training and test sets using `train_test_split()` from `sklearn.model_selection`. The split is performed with a test size of 20% and a random state of 42 to maintain reproducibility.

5) Gaussian Process Regressor

An instance of the GaussianProcessRegressor class is created with an RBF (Radial Basis Function) kernel, which is suitable for capturing smooth patterns. The length scale of the RBF kernel is set to 1.0.

6) Model Fitting

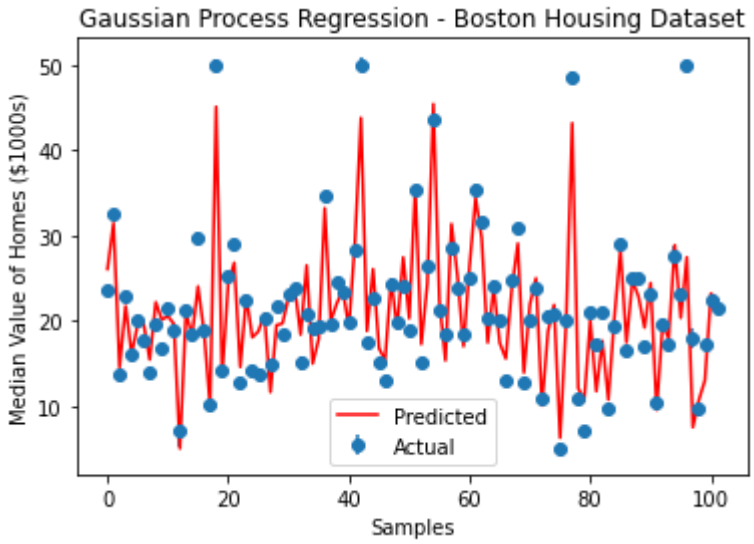
The Gaussian Process regressor is trained on the training data using the `fit()` method. This step estimates the hyperparameters of the chosen kernel based on the provided training samples.

7) Prediction

The trained model is used to predict the median home values for the test data using the `predict()` method. Additionally, the standard deviation of the predictions is computed by setting `return_std=True`.

8) Plotting Results

The predicted values are plotted along with error bars representing the uncertainty in the predictions. The actual target values from the test set are also plotted for comparison. The x-axis represents the samples, and the y-axis represents the median value of homes in thousands of dollars.



The resulting plot provides a visual representation of the Gaussian Process Regression on the Boston Housing dataset. The "Actual" points indicate the true values, and the "Predicted" line represents the model's predictions. The error bars show the uncertainty or confidence intervals around the predictions.

This implementation demonstrates how Gaussian Process Regression can be used to model and predict real estate prices based on different features. The advantage of Gaussian Processes lies in their ability to provide both predictions and associated uncertainties, making them valuable in decision-making scenarios that require understanding prediction confidence.

Further analysis and evaluation could involve performance metrics such as mean squared error or coefficient of determination (R-squared) to assess the model's accuracy and generalization capability. Additionally, different kernel functions and hyperparameter tuning can be explored to optimize the model's performance.