

Nonparametric Regression Project Report



Lecture: CS454/554 - Introduction to Machine Learning
and Artificial Neural Networks

Instructor: Prof. Ethem Alpaydın

Tuna Tuncer – S018474

1. Introduction

I was given one test data set with 100 instances and ten training data sets, each having 25 instances, for this assignment. I was expected to create nonparametric regression models using the formula in figure 1, and then measure and compare the mean squared errors of the models using the training data sets and test data set. It was desired that the nonparametric regression models I would develop had sigma (h) values of 0.05, 0.1, 0.25, 0.5, 1, and 5. The data was generated using a hidden f(x) function and made more random by adding gaussian noise to it.

$$\hat{g}(x) = \frac{\sum_{t=1}^N K\left(\frac{x - x^t}{h}\right) r^t}{\sum_{t=1}^N K\left(\frac{x - x^t}{h}\right)}$$

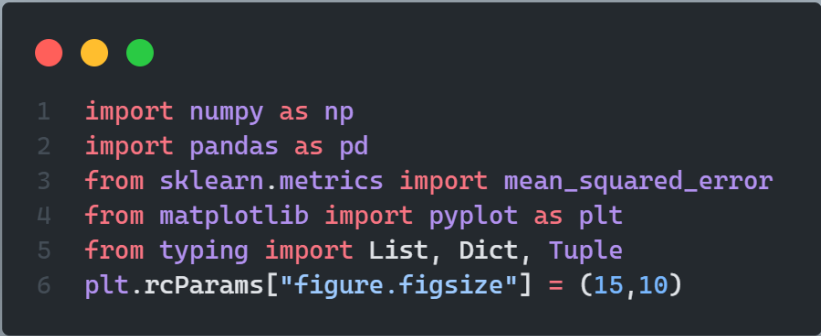
Figure 1: The kernel smoother

2. Methodology

For each sigma value, 10 different models were created, and the mean squared errors of each of these 10 models were calculated. Afterwards, a single value was obtained by averaging 10 different mean squared error values. When this process was repeated for each sigma value, a total of 6 different average mean square error values were obtained. Thus, it has been empirically found that the models with which sigma value explain the data more successfully.

3. Implementation Details

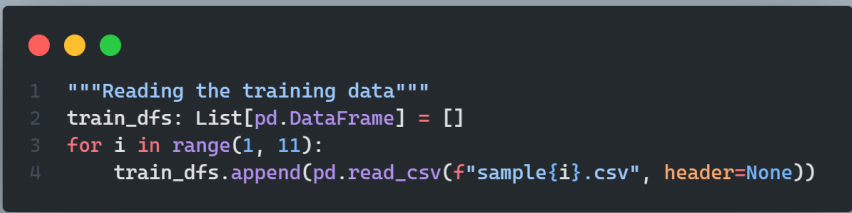
Python is used as the programming language for this assignment. In addition, Jupyter Notebook, which is widely used for machine learning related tasks, is also used as an editor.




```
1 import numpy as np
2 import pandas as pd
3 from sklearn.metrics import mean_squared_error
4 from matplotlib import pyplot as plt
5 from typing import List, Dict, Tuple
6 plt.rcParams["figure.figsize"] = (15,10)
```

Figure [2]: Imported Libraries

As seen in Figure 2, necessary libraries are imported. Pandas is used to read data from CSV files and save it in data frames. NumPy is used to store data in arrays and to take average, and sum of the elements in the array. Sklearn is used to calculate the mean squared error of the models. Matplotlib is used to visualize the entire process. Typing is used to define types for variables throughout the code.



```
1 """Reading the training data"""
2 train_dfs: List[pd.DataFrame] = []
3 for i in range(1, 11):
4     train_dfs.append(pd.read_csv(f"sample{i}.csv", header=None))
```



```
1 """Reading the test data"""
2 test_df: pd.DataFrame = pd.read_csv("test.csv", header=None)
```

Figure [3, 4]: Reading CSV Files and Saving into Data Frames

As seen in Figures 3 and 4, the data were read from the 10 training and 1 test data set CSV files and saved in data frames.

```

1  """Dividing the training data into X and y"""
2  train_dfs_xs: List[np.ndarray] = []
3  train_dfs_ys: List[np.ndarray] = []
4  for train_df in train_dfs:
5      train_dfs_xs.append(train_df.iloc[:, :-1].to_numpy())
6      train_dfs_ys.append(train_df.iloc[:, -1].to_numpy())

```

```

1  """Dividing the test data into X and y"""
2  test_df_xs: np.ndarray = test_df.iloc[:, :-1].to_numpy()
3  test_df_ys: np.ndarray = test_df.iloc[:, -1].to_numpy()

```

Figure [5, 6]: Splitting Input and Output Data

As seen in Figures 5 and 6, the input and output columns in the given training and test data sets are divided into different data frames.

```

1  """Defining the gaussian kernel function"""
2  def gaussian_kernel(u: float) -> float:
3      return np.exp(u ** 2 / -2) / np.sqrt(2 * np.pi)

```

Figure [7]: Implementation of Gaussian Kernel

In Figure 7, gaussian kernel function is implemented because it will be used inside the formula indicated in Figure 1.

```

1  """Write a kernel smoother function which uses the gaussian kernel function"""
2  def kernel_smoother(test_x: float, train_xs: np.ndarray, train_ys: np.ndarray, sigma: float) -> float:
3      """
4      :param test_x: The test point
5      :param train_xs: The training points
6      :param train_ys: The training labels
7      :param sigma: The sigma value
8      :return: The predicted value
9      """
10     numerator: float = np.sum([gaussian_kernel((test_x - train_x) / sigma) * train_y for train_x, train_y in zip(train_xs, train_ys)])
11     denominator: float = np.sum([gaussian_kernel((test_x - train_x) / sigma) for train_x in train_xs])
12     return numerator / denominator

```

Figure [8]: Implementation of the Kernel Smoother

In Figure 8, the kernel smoother is implemented which will be used for the prediction purposes.

```

1  """Write a function which uses the kernel smoother function to predict the test labels"""
2  mse_of_train_dfs_by_sigma: Dict[str, List[Tuple[float, float]]] = {} for i in range(1, 11):
3  # {'train_df number': [(sigma1, mse1), (sigma2, mse2), ...]}
4  sigma_values: List[float] = [0.05, 0.1, 0.25, 0.5, 1, 5]
5
6  """Main algorithm"""
7  for sigma in sigma_values:
8      """Calculating mse values for each sigma value"""
9      for idx, (train_df_xs, train_df_ys) in enumerate(zip(train_dfs_xs, train_dfs_ys), start=1):
10         predictions: List[float] = []
11         for test_df_x, test_df_y in zip(test_df_xs, test_df_ys):
12             prediction: float = kernel_smoother(test_df_x, train_df_xs, train_df_ys, sigma)
13             predictions.append(prediction)
14         error: float = mean_squared_error(test_df_ys, predictions)
15         mse_of_train_dfs_by_sigma[str(idx)].append((sigma, error))
16
17         """Plotting the mse values of every train_dfs for each sigma value"""
18
19         plt.title(f"Plot of g(x)s on 10 different training dfs for h = {sigma}", fontsize=20)
20         plt.plot(test_df_xs, predictions, label=f"Train Df: {idx}")
21         plt.legend(loc='upper left')
22     plt.show()
23

```

Figure [9]: The Main Algorithm

In Figure 9, gaussian kernel and kernel smoother functions are used to get predictions and these predictions are then used to calculate the mean squared error. Afterwards, 6 different plots are generated for 6 different sigma values, and every plot contains the predictions of 10 different models on the test dataset.

```

1  mse_values_by_sigma: Dict[float, List[float]] = {
2      sigma: [] for sigma in sigma_values
3  }
4
5  for sigma_mse_list in mse_of_train_dfs_by_sigma.values():
6      for sigma, mse in sigma_mse_list:
7          mse_values_by_sigma[sigma].append(mse)
8
9  plt.title(f"Plot of Average MSE values for different sigma values", fontsize=20)
10 plt.bar(range(6), [np.mean(mses) for mses in mse_values_by_sigma.values()])
11 plt.xticks(range(6), sigma_values)
12 plt.xlabel('Sigma (h) value', fontsize=15)
13 plt.ylabel('Average MSE', fontsize=15)
14 plt.show()

```

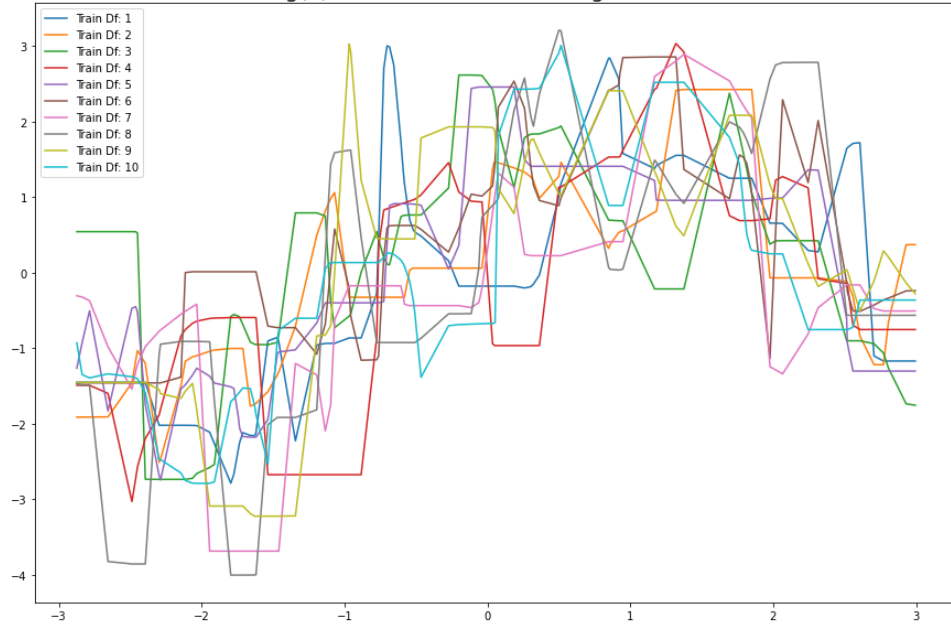
Figure [10]: Mean Squared Errors for All Models of All Sigma Values

Finally in Figure 10, the average mean squared errors are calculated and plotted for every different sigma value.

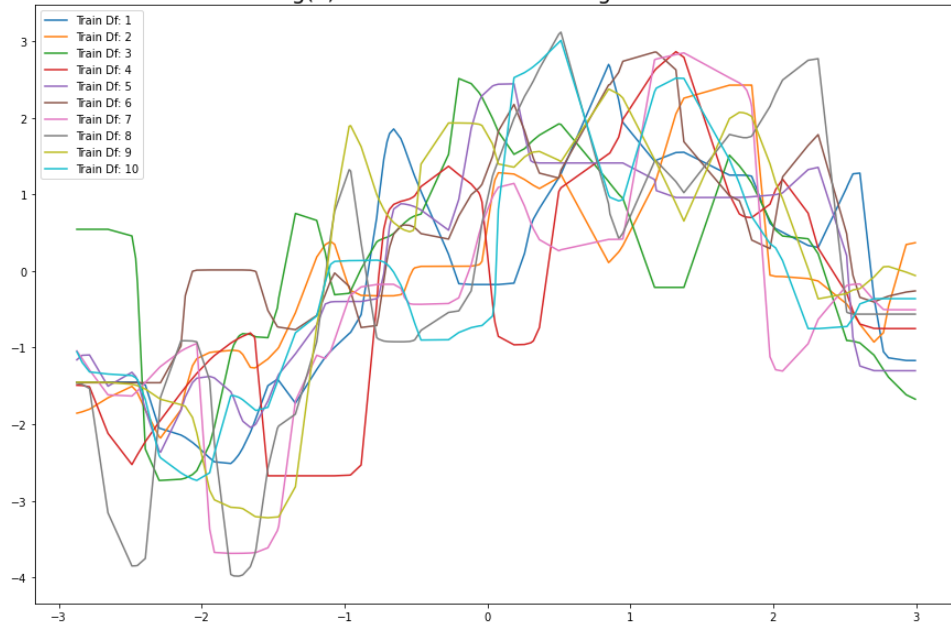
4. Results

Below you can see the plots of 10 different predictions and average mean squared errors on every sigma value.

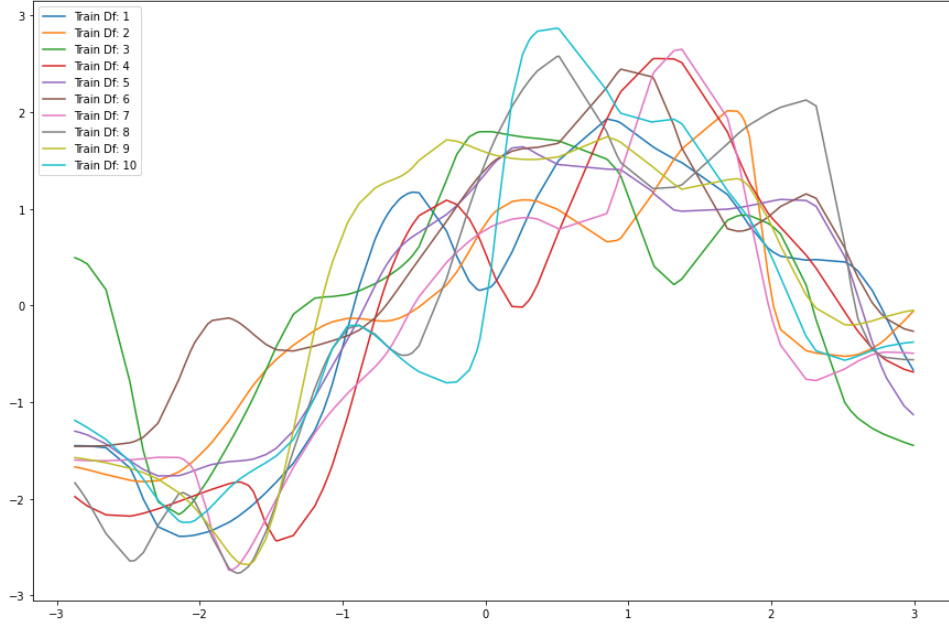
Plot of $g(x)$ s on 10 different training dfs for $h = 0.05$



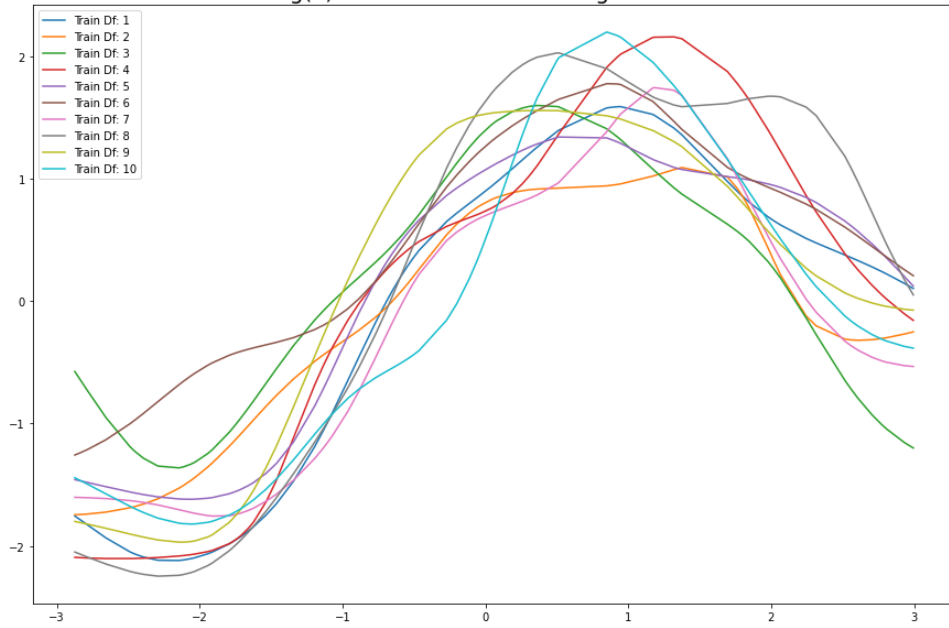
Plot of $g(x)$ s on 10 different training dfs for $h = 0.1$



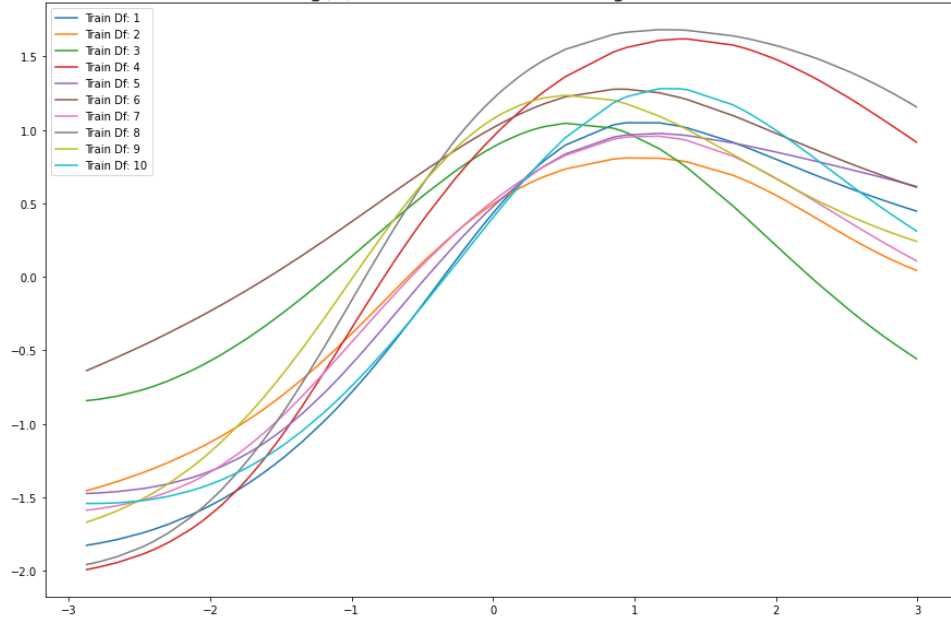
Plot of $g(x)$ s on 10 different training dfs for $h = 0.25$



Plot of $g(x)$ s on 10 different training dfs for $h = 0.5$



Plot of $g(x)$ s on 10 different training dfs for $h = 1$



Plot of $g(x)$ s on 10 different training dfs for $h = 5$

