“In The Name Of God”

HW#3 Report

Professor: Dr.Azmifar

Presenters:

Narges Dehghan

Masome Jafari

Table of Contents

Abstract

This assignment encompasses a comprehensive exploration of implement non-parametric density estimation methods and how to visualize the resulted probability density functions., fundamental techniques in the realm of machine learning and data analysis. The primary objectives are twofold: to understand the fundamental concepts and application of these methods and to apply this knowledge to real-world datasets.

Non-Parametric Density Estimation

Histograms

In this code analyzes a 1D dataset ('1D_grades.csv') by creating histograms with varying bandwidths (0.1, 0.3, 0.5, 0.7). Additionally, the code incorporates a probability density function (PDF) scatter plot on each histogram to visualize the distribution of normalized grades.

Data Loading and Normalization:

The dataset is loaded using pandas, and the values are normalized by subtracting the mean and dividing by the standard deviation.

Bandwidth Variation:

The code iterates through bandwidth values of 0.1, 0.3, 0.5, and 0.7.

Histogram Calculation:

For each bandwidth, the minimum and maximum values for the x-axis are determined.

Frequencies are calculated for each bin, and histograms are plotted using matplotlib.

PDF Scatter Plot:

A density function (dens) is introduced to compute the PDF at specific points within each bandwidth.

A scatter plot of the PDF is added to each histogram subplot.

Visualization:

Subplots are created for each bandwidth, with histograms and PDF scatter plots.

The layout is adjusted for clarity, and the final visualization is displayed.

Results and Insights:

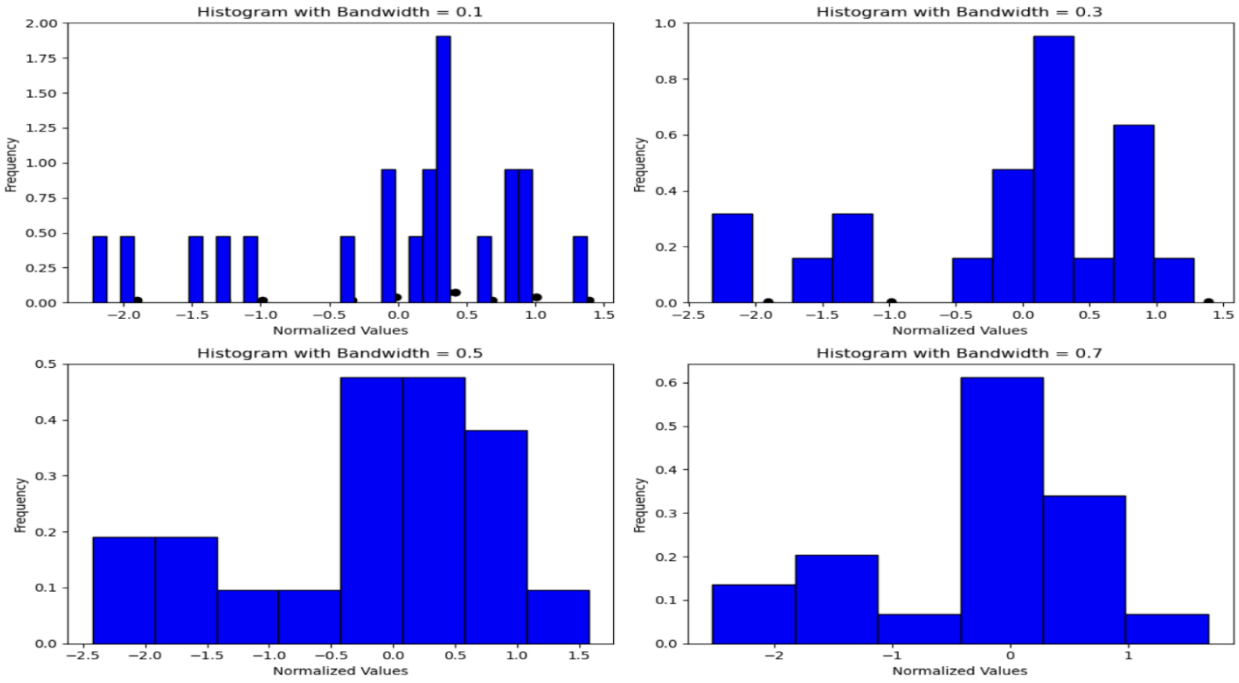
The generated plots provide a comprehensive overview of the dataset's distribution under different bandwidths.

The PDF scatter plots offer insights into the probability density at specific points, enhancing the understanding of the data's characteristics.

Varying bandwidths allow for exploration of different levels of granularity in the data representation.

Conclusion:

The code successfully creates informative visualizations, combining histograms and PDF scatter plots to enhance the analysis of a 1D dataset with varying bandwidths. This approach facilitates a nuanced exploration of data distribution, aiding in the identification of patterns and insights.



In this code aims to visualize a 2D dataset ('2D_synthetic_gaussians.csv') by generating 3D bar plots with varying bandwidths (0.1, 0.3, 0.5, 0.7). The code utilizes a manual calculation for creating the 2D histogram and incorporates probability density function (PDF) values for representing the joint distribution of the two variables.

Code Overview:

Data Loading and Normalization:

The dataset is loaded using pandas, and values are normalized by subtracting the mean and dividing by the standard deviation.

Bandwidth Variation:

The code iterates through bandwidth values of 0.1, 0.3, 0.5, and 0.7.

Manual Histogram Calculation:

Instead of using np.histogram2d, the code manually calculates the 2D histogram by looping through the data and incrementing the corresponding bins.

PDF Calculation:

A density function calculates the PDF values for each point within the 2D dataset, enhancing the representation accuracy.

3D Bar Plotting:

3D bar plots are created using ax.bar3d from the mpl_toolkits.mplot3d module.

The positions (xpos, ypos) and dimensions (dx, dy, dz) of the bars are defined based on the PDF values.

Visualization:

The code generates a 2x2 grid of 3D bar plots, each representing the joint distribution with a different bandwidth.

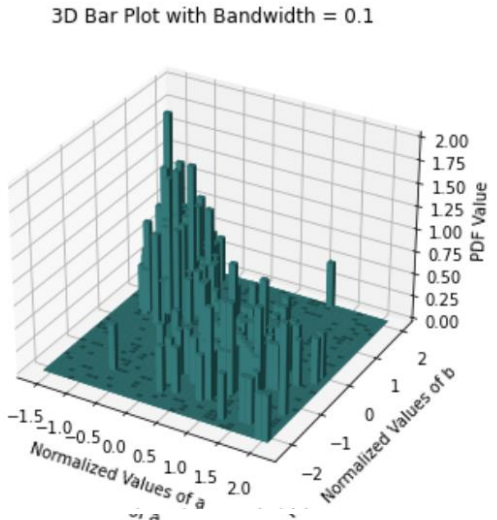
PDF values are used to provide a more continuous and accurate representation of the dataset.

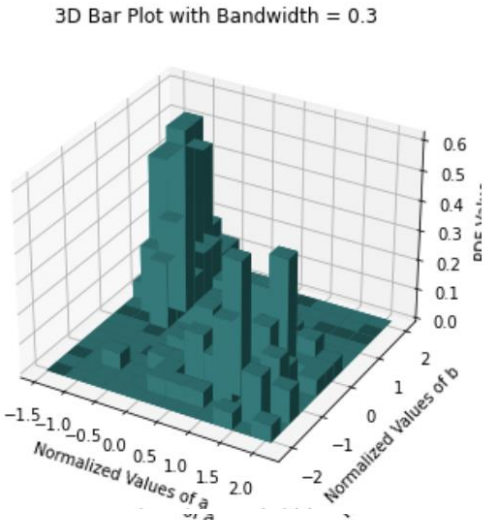
The manual histogram calculation offers a straightforward approach, providing transparency into the creation of the 2D histogram.

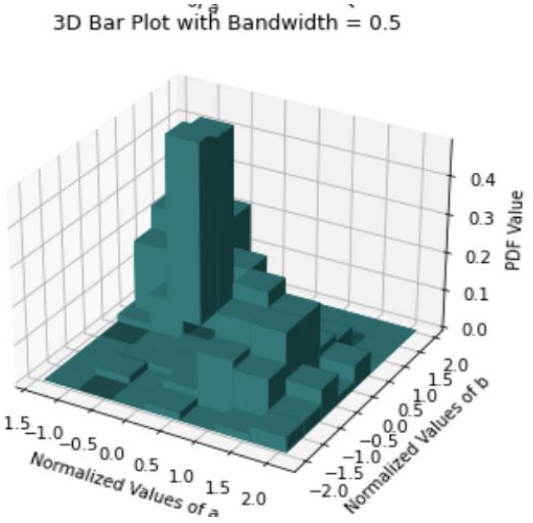
The use of PDF values in the 3D bar plots enhances the visual representation of the joint distribution, especially in regions with sparse data points.

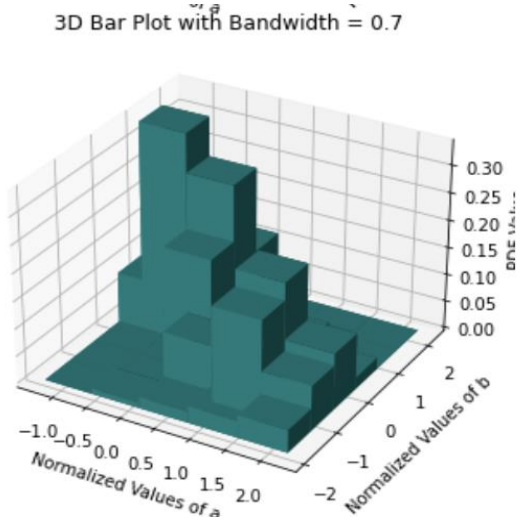
Conclusion:

The code successfully achieves the visualization of 2D dataset distribution through 3D bar plots with varying bandwidths. The manual histogram calculation and inclusion of PDF values contribute to a more accurate and insightful representation of the dataset's joint distribution.

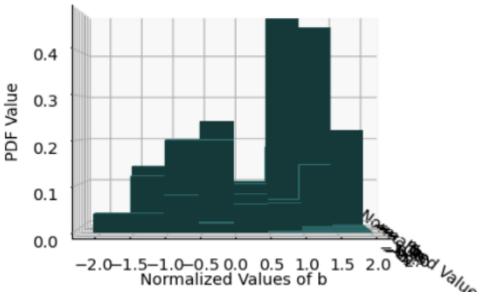




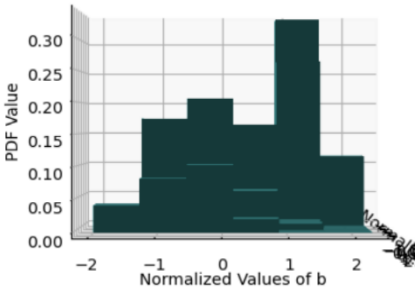




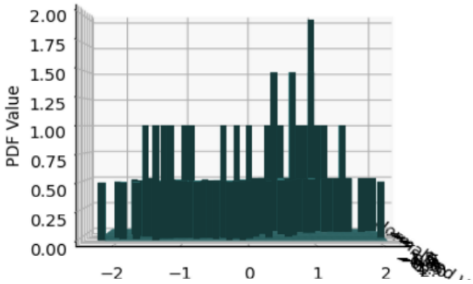
3D Bar Plot with Bandwidth = 0.5



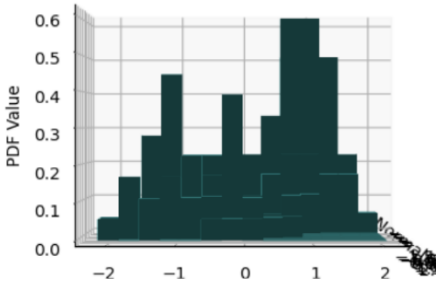
3D Bar Plot with Bandwidth = 0.7



3D Bar Plot with Bandwidth = 0.1



3D Bar Plot with Bandwidth = 0.3



Kernels

KDE

The code provided Kernel Density Estimation (KDE) using both the Parzen window and Gaussian kernel. Here's a breakdown of the key components.Parzen Window and Gaussian Kernel Functions:The parzenWindowd(u) function represents the Parzen window kernel.The gaussian(u) function represents the Gaussian kernel.

KDE Function:

The KDE function computes the Kernel Density Estimation using a specified kernel type (either 'parzen' or 'gaussian').It iterates over each point in the 2D input space (X_2d) and calculates the probability density for that point based on the chosen kernel type.

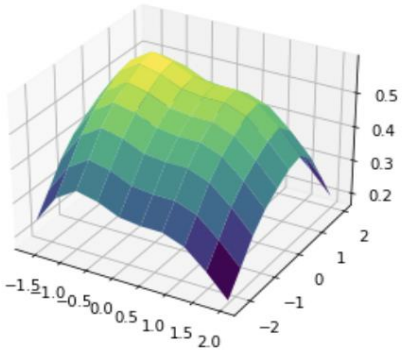
Visualization:

For each bandwidth value in Hs, the code calculates KDE and visualizes the results.The 3D surface plots using ax.plot_surface represent the KDE using Gaussian kernel with varying bandwidths.The contour plots using ax.contour provide additional visualization of the KDE using both Gaussian and Parzen window kernels.

Conclusion:

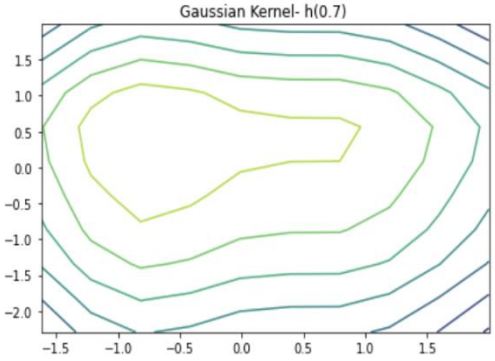
The code effectively demonstrates Kernel Density Estimation using both Parzen window and Gaussian kernel functions.It visualizes the KDE results with 3D surface plots and contour plots, offering insights into the estimated probability density distribution for different bandwidths. Also, separate code for a 1D data set.

Gaussian Kernel - h(0.7)

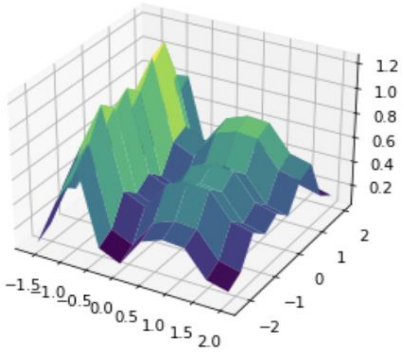


<Figure size 432x288 with 0 Axes>

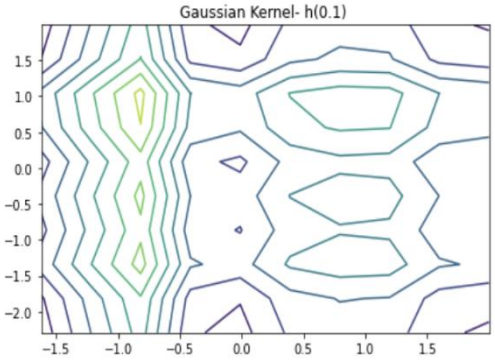
Gaussian Kernel- h(0.7)



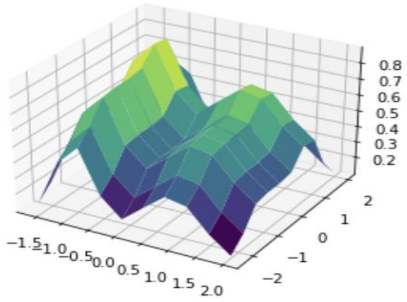
Gaussian Kernel - h(0.1)



Gaussian Kernel- h(0.1)

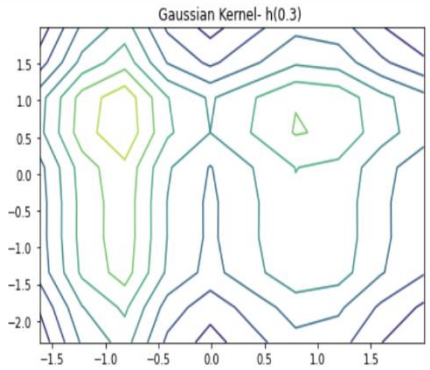


Gaussian Kernel - h(0.3)

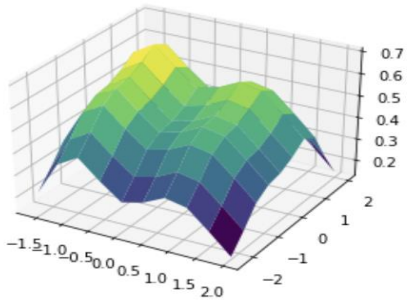


<Figure size 432x288 with 0 Axes>

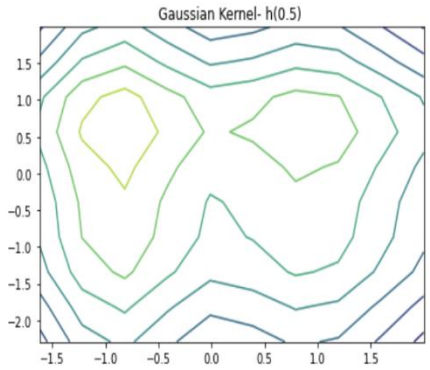
Gaussian Kernel- h(0.3)

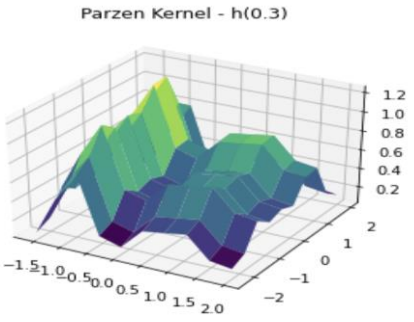


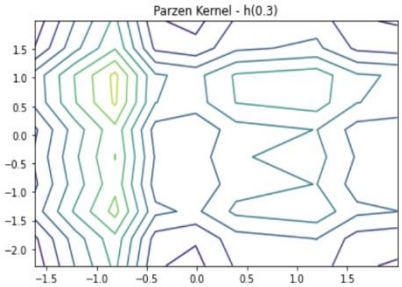
Gaussian Kernel - h(0.5)

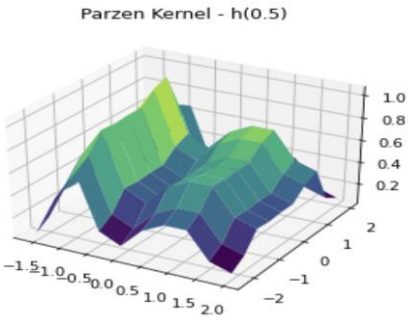


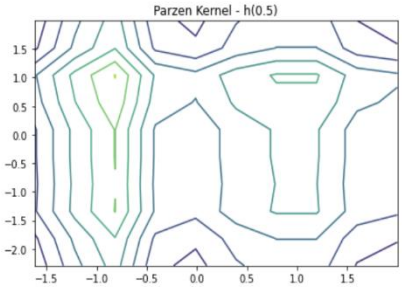
Gaussian Kernel- h(0.5)

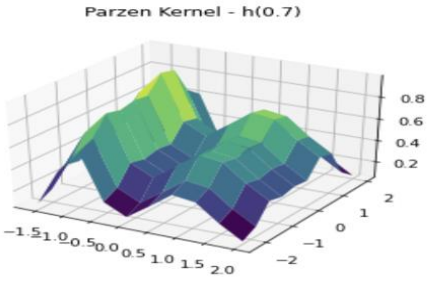


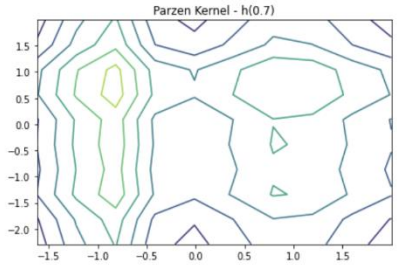

Parzen Kernel - h(0.3)

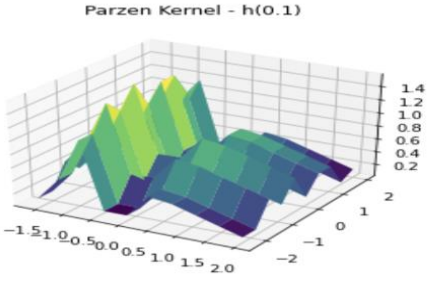

Parzen Kernel - h(0.3)

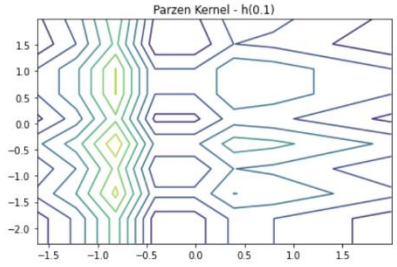

Parzen Kernel - h(0.5)

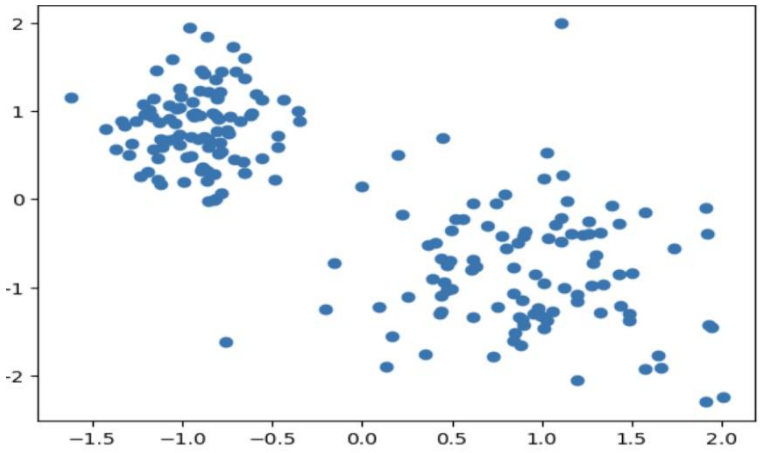

Parzen Kernel - h(0.5)

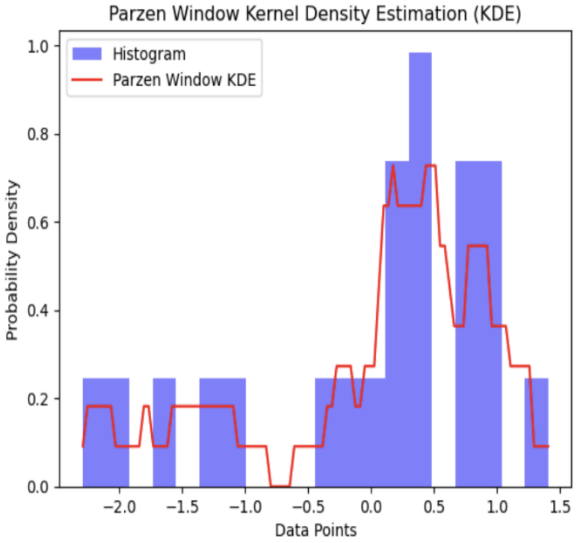

Parzen Kernel - h(0.7)

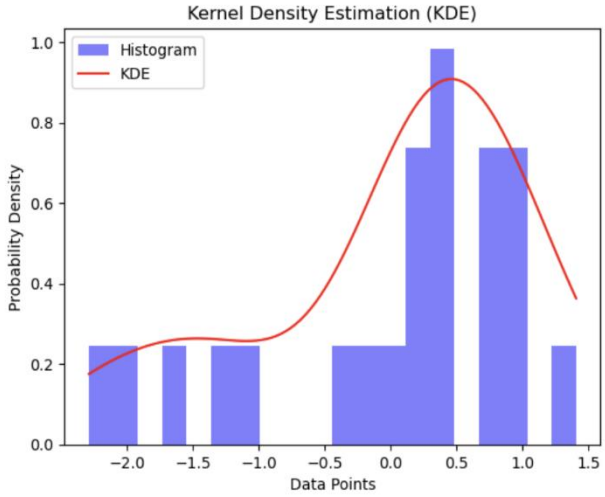

Parzen Kernel - h(0.7)

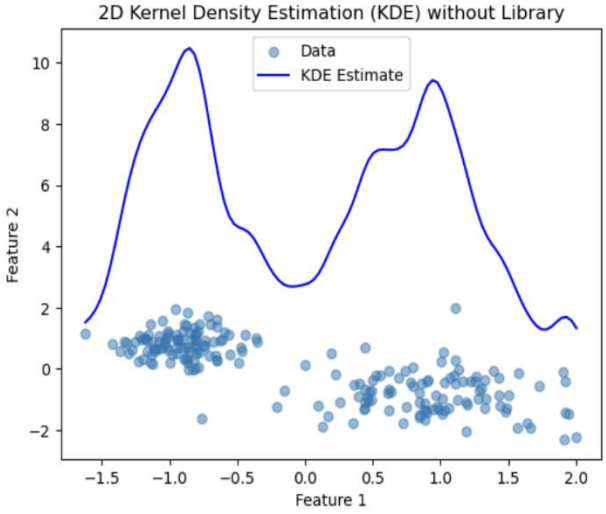

Parzen Kernel - h(0.1)


Parzen Kernel - h(0.1)




Parzen Window Kernel Density Estimation (KDE)
Histogram
Parzen Window KDE
Probability Density
Data Points


Kernel Density Estimation (KDE)
Histogram
KDE
Probability Density
Data Points


2D Kernel Density Estimation (KDE) without Library
Data
KDE Estimate
Feature 2
Feature 1

KNN

Thi code aims to perform Probability Density Estimation using the K-Nearest Neighbors (KNN) method. It calculates the probability density at each point in a 2D space for different values of k.

KNN Functions:

getDistance: Calculates the distance between a point and its k-1 nearest neighbors.

KNN: Computes the probability density using the KNN method for each point in the 2D space.

Data Preparation and Meshgrid Creation:

Meshgrid (xx, yy) is created for the 2D input space using numpy. Probability density is calculated for each point in the meshgrid using KNN with different values of k.

Visualization:

3D surface plots are generated using ax.plot_surface for each value of k.

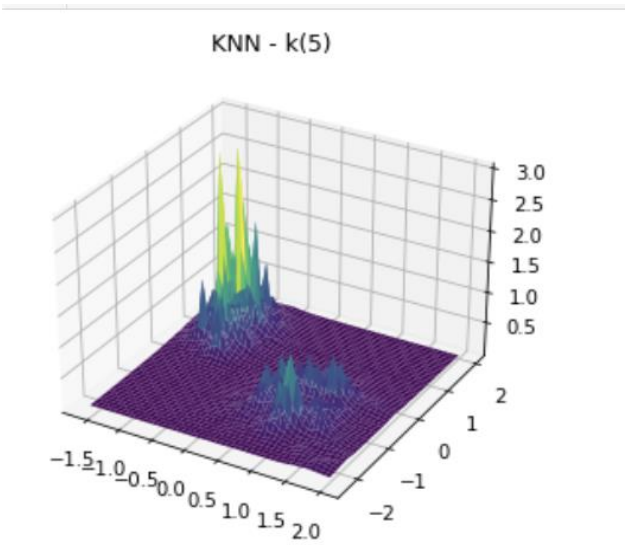
Each plot represents the estimated probability density using KNN for varying numbers of nearest neighbors.

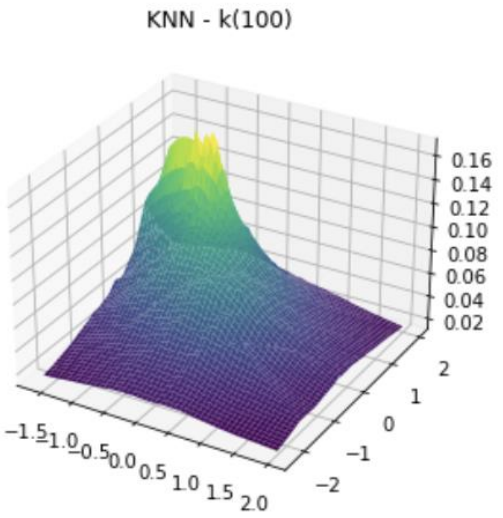
The code successfully applies the KNN method for probability density estimation on a synthetic 2D dataset. The distance metric used for KNN is the Euclidean distance. Different values of k (5, 50, 100, 200) are explored to observe the impact on the estimation.

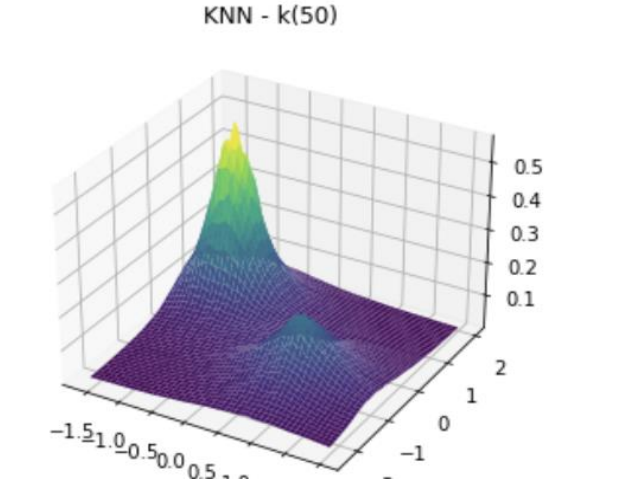
Conclusion:

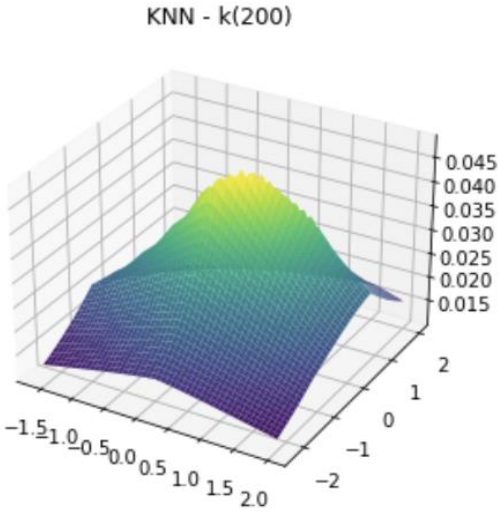
The code provides an effective implementation of K-Nearest Neighbors for Probability Density Estimation in a 2D space.

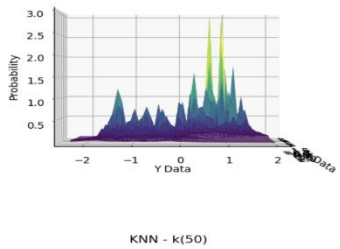
The visualization of estimated probability density surfaces allows for a better understanding of how the choice of k influences the method's performance.

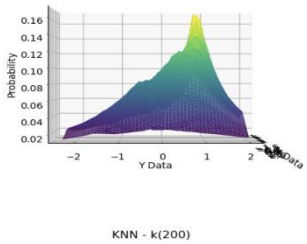

KNN - k(5)
3.0
2.5
2.0
1.5
1.0
0.5
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0
2
1
0
−1
−2

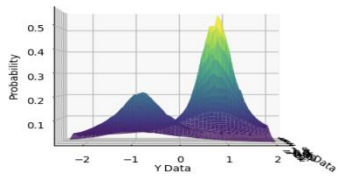

KNN - k(100)
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0
2
1
0
−1
−2

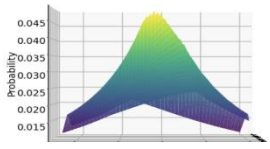

KNN - k(50)
0.5
0.4
0.3
0.2
0.1
−1.5 −1.0 −0.5 0.0
2
1
0
−1

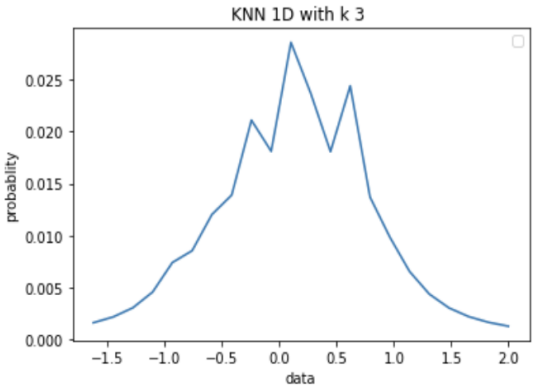

KNN - k(200)
0.045
0.040
0.035
0.030
0.025
0.020
0.015
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0
2
1
0
−1
−2

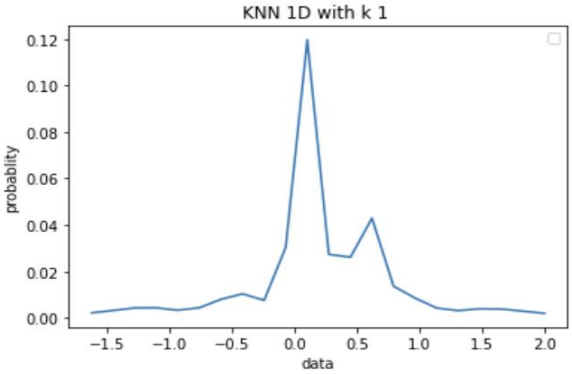

Probability
Y Data
KNN - k(50)

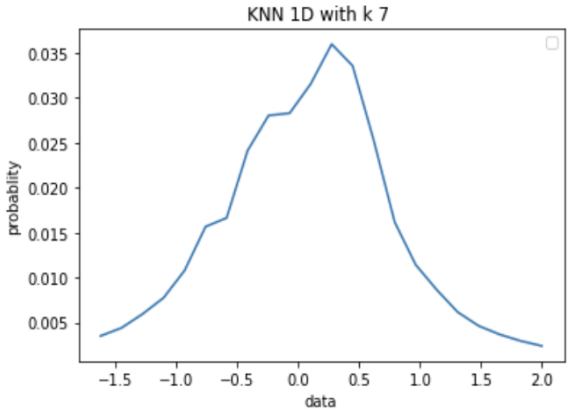

Probability
Y Data
KNN - k(200)

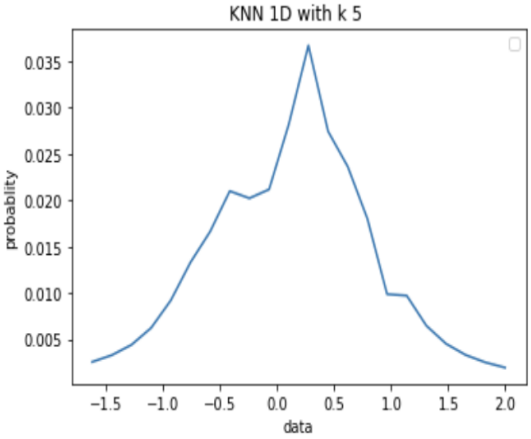

Probability
Y Data


Probability


KNN 1D with k 3
probablity
data


KNN 1D with k 1
probablity
data


KNN 1D with k 7
probablity
data


KNN 1D with k 5
probablity
data

Conclusion

Both codes serve as educational examples of how to implement and apply fundamental machine learning algorithms. They highlight the importance of data preprocessing, parameter optimization, and result visualization in the context of regression and classification tasks. These codes can be a starting point for more advanced modeling and analysis of the respective domains they address.