Nonparametric Approaches
Assignment 3

Instructor	Iason Oikonomidis	oikonom@csd.uoc.gr
T.A.	Giorgos Karvounas	gkarv@csd.uoc.gr

Deadline **Thursday 23:59, 18th of April 2024**

Abstract

The goal of this assignment is some hands-on experience with nonparametric density estimation and classification techniques. Specifically, you will experiment with Parzen windows and k Nearest Neighbors on some provided data.

Note: This is a personal assignment and should be pursued individually and without use of any computerized AI facilities. Note that there will be a face-to-face examination after the delivery. The assignment should be implemented entirely in Google Colaboratory following the deliverable instructions below. *Use library implementations only when explicitly stated*.

Question A: Parzen Windows (55/100)

The scope of this question is to get familiar with the Parzen Windows density estimation approach. Towards this goal, you are asked to estimate the probability density function of the distribution of some given data. **Data**: you will use the dataset included in the *dataA_Parzen.csv* file. The dataset consists of 200 samples, as rows, and each sample is 1-dimensional (overall, 1 column). We do not have any knowledge or assumption about their distribution.

1. Implement the `hypercube` window function φ(u), when the window is a hypercube. You may use the `numpy.abs` function. The data is 1-dimensional, but try to implement the general hypercube. (no penalty if it works only for d=1).
2. Implement the `gaussian` window function φ(u), a Gaussian kernel. You may use the `numpy.exp` function. Again, it is recommended (but not graded) to implement the general case for d-dimensional input.
3. Implement a function that takes as input a single point `x`, the center of the window as a single point `c`, the width `h` of the window and the kernel type (`'hypercube'` or `'Gaussian'`) of the window. The function is responsible to call one of the above implemented functions (hypercube or gaussian), with the appropriate input, and return the result.
4. Define the `density_parzen` function, implementing the Parzen Window density estimation approach. The function should take as input an array of 1-d points `data`, a single point `x` which represents the center, the width `h` of the window and the kernel type of the window. The function should return the likelihood of `x`, given the other inputs.
5. What's the best value for the width of the window `h`? Assume that the dataset you have comes from the normal distribution *N(0, 4)*, a univariate normal distribution with μ=0 and σ^2=4. Given this, to find the optimal value for `h`:

   (a) Create a histogram of the data to convince yourselves that they come from the aforementioned distribution. You may use the `matplotlib.pyplot.hist` function here.

   (b) For `h` in the range `[0.2, 10]` with step = `0.1` calculate: 1) the predicted likelihood for every point in the data, 2) their true likelihood (use the function `scipy.stats.norm.pdf(data, loc=0, scale=2)`), and 3) the Mean Square Error (MSE) of the two likelihoods, estimated and true, over all samples. The MSE between two arrays of values in one-to-one correspondence, $\overline{vals}$ (estimated) and $vals$ (ground truth), can be computed using the snip: `sum( (vals_bar[i] - vals[i]) ** 2 for i in range(n) ) / n`, with `n = len(vals)`. Repeat this process for both kernels (hypercube and Gaussian). What's the best value for `h` for each kernel? Print your answers and create a plot with the values of `h` on the x-axis and the MSE on the y-axis for both kernels, showing two curves overall.

Release date: Tuesday, 3rd of April 2024
Deadline: Thursday 23:59, 18th of April 2024

Question B: K-Nearest Neighbors (KNN) Classification (55/100)

The goal of this question is to get hands-on experience with the K-Nearest Neighbors (KNN) Classifier. **Data**: you will use the dataset included in the dataB_KNNtrain.csv and dataB_KNNtest.csv files. Each dataset consists of 50 samples, as rows. You can confirm that each sample has 3 columns. The first 2 columns represent the 2-dimensional data and the last column represents their label, in {0, 1}. You should use the labels of the test data as ground truth for computation of the achieved accuracy of the classifier you will develop.

1. Implement a function that takes as input a 2D point x and a NumPy array training dataset of 2D points, and computes the Euclidean distances of the point x to all points in the given array. The function should return a NumPy array of the computed Euclidean distances. Here, you should implement the euclidean distance, don't use `np.norm` – but you can use exponentiation such as squaring and taking the square root using NumPy as needed.
2. Implement a function which takes as input a 2D point x, a NumPy array training dataset of 2D points and a number k. The function returns the k closest neighbors of x among the training data, and their indices. You may use/call the function from the previous Question. You are not asked to implement sorting here, use library calls as needed.
3. In this step you are asked to develop the k-NN algorithm. Implement a function `knn_classify` which takes as input the training data, the test data and a number k of neighbors that will be considered during k-NN. The function should return two probabilities for each sample x_i of the test data, the probability of sample x_i belonging to class 0 and the probability of sample x_i belonging to class 1, respectively. These probabilities should add to 1. In this step you should carefully handle the 2D features (first two columns), separately from the label (last column).
4. In this question, you are asked to search for the best values of the (meta)parameter k. Specifically, find the value of k that maximizes the accuracy of the k-NN classifier. Compute the accuracy of the classifier from Question 3, for each k in the set of 1,3,5,7, .. ,15. Plot with point markers the results, print and explain which k you would choose.
5. **Optionally**: Use the code below to show the decision boundaries of your classifier in a 2D plot, using the value for k from the previous task. Assumptions: `optimal_k` is the value computed in the previous question and `train_data` is the array of the data in the dataB_KNNtrain.csv file.

```
x_min, x_max = 2, 25
y_min, y_max = 15, 45
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1

 Z = np.array([knn_classify(train_data, np.array([x, y]), optimal_k)[1]
               for x, y in zip(xx.ravel(), yy.ravel())])

# comment out the line below to show probabilities instead of decision regions
Z = (Z > 0.5).astype(int)
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.5, cmap=plt.cm.coolwarm)
plt.scatter(train_data[:, 0], train_data[:, 1],
          c=train_data[:, 2], cmap=plt.cm.coolwarm, edgecolors='k')
plt.set_aspect(1)
plt.show()
```

Release date: Tuesday, 3rd of April 2024
Deadline: Thursday 23:59, 18th of April 2024

Deliverable

This assignment should be implemented entirely in Google Colaboratory. Your deliverable should be a single .ipynb file (can be easily exported from Google Colaboratory). Every single question should be implemented in a single code block. Code blocks should be clearly and shortly explained (you may use the text boxes for that goal). **Only use library functions for arithmetic operations, matrix operations and plots, unless explicitly stated.**

Release date: Tuesday, 3rd of April 2024
Deadline: Thursday 23:59, 18th of April 2024