Parzen with soft windows (kernels)

In this Homework we will use Iris as a toy dataset. It contains 150 samples (one for each row), each with 4 features (the 4 first columns) and one label in {1,2,3} (the 5th column). It is recommended you download it here and then test your code by importing it like this :

```
import numpy as np
iris = np.loadtxt('iris.txt')
```

When the answer template in solution.py has "iris" as an argument, you may assume that this argument is the dataset in numpy format. Your function should use this argument to perform computations, not a version of the dataset that you loaded by yourself.

1. Undergraduates 4 pts Graduates 4 pts
   Question. Write functions that take that dataset as input and return the following statistics:

   (a) Q1.feature_means : An array containing the empirical means of each feature, from all examples present in the dataset. Make sure to maintain the original order of features.
   e.g. : Q1.feature_means(iris) = $[\mu_1, \mu_2, \mu_3, \mu_4]$

   (b) Q1.covariance_matrix : A 4×4 matrix that represents the empirical covariance matrix of features on the whole dataset.

   (c) Q1.feature_means_class_1 : An array containing the empirical means of each feature, but only from examples in class 1. The possible classes in the iris dataset are 1, 2, and 3.
   e.g. : Q1.feature_means_class_1(iris) = $[\mu_1, \mu_2, \mu_3, \mu_4]$

   (d) Q1.covariance_matrix_class_1 : A 4×4 matrix that represents the empirical covariance matrix of features, but only from examples in class 1.

2. Undergraduates 1 pts Graduates 1 pts
   Question. Implement Parzen with hard window parameter h. Use the standard Euclidean distance on the original features of the dataset. Your answer should have the following behavior :
   f = HardParzen(h) initiates the algorithm with parameter h
   f.train(X, Y) trains the algorithm, where X is a $n \times m$ matrix of n training samples with m features, and Y is an array containing the n labels. The labels are denoted by integers, but the number of classes in Y can vary.
   f.compute_predictions(X_test) computes the predicted labels and return them as an array of same size as X_test. X_test is a $k \times m$ matrix of k test samples with same number of features as X. This function is called only after training on (X, Y). If a test sample x has no neighbor within window h, the algorithm should choose a label at random by using **draw_rand_label(x, label_list)**, a function that is provided in the **solution.py** file, where label_list is the list of different classes present in Y, and x is the array of features of the corresponding point.

3. Undergraduates 10 pts Graduates 5 pts
   Question. Implement Parzen with a soft window. We will use a radial basis function (RBF) kernel (also known as *Gaussian* kernel) with parameter σ. Use the standard Euclidean distance on the original features of the dataset. Please refer to the slides from the second week for the definition. The structure of your solution should be the same as in the previous question, but you will never need to draw a label at random with **draw_rand_label(x, label_list)**. The class name is **SoftRBFParzen**.

4. Undergraduates 7 pts Graduates 5 pts
   Question. Implement a function **split_dataset** that splits the Iris dataset as follows:

   - A training set consisting of the samples of the dataset with indices which have a remainder of either 0, 1, or 2 when divided by 5
   - A validation set consisting of the samples of the dataset with indices which have a remainder of 3 when divided by 5.
   - A test set consisting of the samples of the dataset with indices which have a remainder of 4 when divided by 5.

For instance the element of index 14 (in the original dataset) should be part of the test set because the remainder of 14 divided by 5 is 4. Do not use random splitting for this exercise (even though it is generally a very good idea). The function should take as input the dataset and return the three sets as a tuple (train, validation, test), where each element of the tuple is a matrix with 5 columns (the 4 features and the labels, kept in the same order).

5. Undergraduates 15 pts Graduates 10 pts

   Question. Implement two functions **ErrorRate.hard_parzen** and **ErrorRate.soft_parzen** that compute the error rate (i.e. the proportion of missclassifications) of the HardParzen and SoftRBFParzen algorithms. The expected behavior is as follows :
   test_error = ErrorRate(x_train, y_train, x_val, y_val) initiates the class and stores the training and validation sets, where x_train and x_val are matrices with 4 feature columns, and y_train and y_val are arrays containing the labels.
   test_error.hard_parzen(h) takes as input the window parameter h and returns as a float the error rate on x_val and y_val of the HardParzen algorithm that has been trained on x_train and y_train.
   test_error.soft_parzen(σ) works just like with Hard Parzen, but with the SoftRBFParzen algorithm.

   Then, include in your report a single plot with two lines:

   (a) Hard Parzen window's classification error on the validation set of iris, when trained on the training set (see question 4) for the following values of h:

   $$h \in \{0.001, 0.01, 0.1, 0.3, 1.0, 3.0, 10.0, 15.0, 20.0\}$$

   (b) RBF Parzen's classification error on the validation set of iris, when trained on the training set (see question 4) for the following values of σ:

   $$\sigma \in \{0.001, 0.01, 0.1, 0.3, 1.0, 3.0, 10.0, 15.0, 20.0\}$$

   The common x-axis will represent either h or σ. Always label your axes and lines in the plot!

   Give a detailed discussion of your observations.

6. Undergraduates 6 pts Graduates 5 pts

   Question. Implement a function **get_test_errors** that uses the evaluated validation errors from the previous question to select h^* and σ^*, then computes the error rates on the test set. The value h^* is the one (among the proposed set in question 5) that results in the smallest validation error for Parzen with hard window, and σ^* is the parameter (among the proposed set in question 5) that results in the smallest validation error for Parzen with RBF.

   The function should take as input the dataset and split it using question 4. The expected output is an array of size 2, the first value being the error rate on the test set of Hard Parzen with parameter h^* (trained on the training set), and the second value being the error rate on the test set of Soft RBF Parzen with parameter σ^* (trained on the training set).

7. Undergraduates 6 pts Graduates 5 pts

   Question. Include in your report a discussion on the running time complexity of these two methods. How does it vary for each method when the hyperparameter h or σ changes? Why ?

8. Undergraduates [bonus] pts Graduates 5 pts

   Question. Implement a random projection (Gaussian sketch) map to be used on the input data:

   Your function **random_projection** should accept as input a feature matrix X of dimension $n \times 4$, as well as a 4×2 matrix A encoding our projection.

   Define $p : x \mapsto \frac{1}{\sqrt{2}} A^T x$ and use this random projection map to reduce the dimension of the inputs (feature vectors of the dataset) from 4 to 2.

   Your function should return the output of the map p when applied to X, in the form of a $n \times 2$ matrix.
   e.g. $project\_data(X_{n\times4}, A_{4\times2}) = X^{proj}_{n\times2}$

9. Undergraduates [bonus] pts Graduates 10 pts

   Question. Similar to Question 5, compute the validation errors of Hard Parzen classifiers trained on 500 random projections of the training set, for

$$h \in \{0.001, 0.01, 0.1, 0.3, 1.0, 3.0, 10.0, 15.0, 20.0\}$$

The validation errors should be computed on the projected validation set, using the same matrix A. To obtain random projections, you may draw A as 8 independent variables drawn uniformly from a gaussian distribution of mean 0 and variance 1.
You can for example store these validation errors in a 500×9 matrix, with a row for each random projection and a column for each value of h.

Do the same thing for RBF Parzen classifiers, for

$$\sigma \in \{0.001, 0.01, 0.1, 0.3, 1.0, 3.0, 10.0, 15.0, 20.0\}$$

Plot and include in your report in the same graph the average values of the validation errors (over all random projections) for each value of h and σ, along with error bars of length equal to $0.2\times$ the standard deviations.

How do your results compare to the previous ones?