IFT 6390 Fundamentals of Machine Learning
Ioannis Mitliagkas

Homework 1 - Practical component

- This homework must be done and submitted to Gradescope individually. You are welcome to discuss with other students but the solution and code you submit must be your own. Note that we will use Gradescope's plagiarism detection feature. All suspected cases of plagiarism will be recorded and shared with university officials for further handling.
- The practical part should be coded in python (you can use the numpy and matplotlib libraries) and all code will be submitted as a python file to Gradescope. To enable automated code grading you should work off of the template file given in this homework folder. Do not modify the name of the file or any of the function signatures of the template file or the code grading will not work for you. You may, of course, add new functions and any regular python imports.
- Any graphing, charts, or practical report parts should be submitted as a pdf file to the Gradescope entry `HW1 Practical Report`. For the report it is recommended to use a Jupyter notebook, writing math with MathJax and export as pdf. You may alternatively write your report in LaTeX; LyX; Word. In any case, you should export your report to a pdf file that you will submit. You are of course encouraged to draw inspiration from what was done in lab sessions.

You should work off of the template file `solution.py` in the project and fill in the basic numpy functions using numpy and python methods

Parzen with soft windows (kernels)

In this Homework we will use banknote authentication Data Set as a toy dataset. It contains 1372 samples (one for each row), each with 4 features

(the 4 first columns) and one label in {0,1} (the 5th column). It is recommended you download it here and then test your code by importing it like this :

```
import numpy as np
banknote = np.genfromtxt('data_banknote_authentication.txt',
                         delimiter=',')
```

When the answer template in solution.py has "banknote" 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. [4 points]
   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(banknote) = [μ_1,μ_2,μ_3,μ_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 banknote dataset are 0 and 1.
   e.g. : Q1.feature_means_class_1(banknote) = [μ_1,μ_2,μ_3,μ_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. [1 points]
   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. [5 points]
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. [5 points]
Question. Implement a function **split_dataset** that splits the banknote dataset as follows:

   - A training set consisting of the samples of the dataset with indices which have a remainder of either 0 or 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. [10 points]
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 banknote, when trained on the training set (see question 4) for the following values of h:

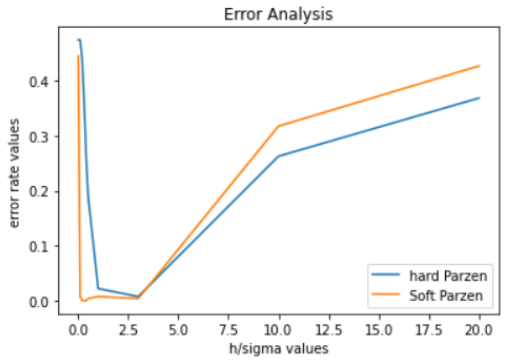
$$h \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0\}$$

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

$$\sigma \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0\}$$

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.
ANS:



I infer Bias Variance Trade-off in the error Analysis plot.

- In terms of hard Parzen, for h=0.01, there was high error rate as the model is not complex enough to capture the data points perfectly. As the h value slightly increase, the error rate drops drastically from 0.48 to around 0.05. As the h value increased further, there is a steep increase in the error rate after the h value greater than 3. So, from this graph the ideal rate to choose h would be around 3 with which we can get a better error rate with good model complexity so that our model learns rather than under-fit or over-fit.
- In terms of Soft Parzen, for sigma=0.01 similar to hard Parzen, the model underfits and the bias to much higher as the model couldn't perfectly catch the points in the data. However, as the sigma value is slightly increased, the error is almost 0 for certain sigma values. At certain point, the error rates in soft Parzen window method sky rocketed for higher sigma values.

When we compare Hard and Soft Parzen error graph, the soft Parzen converged and provides slightly better error rate than Hard Parzen method. However, if the sigma values go up, if performs poorly compared to the hard Parzen. In my opinion, using Soft Parzen is preferred in terms of error rate. Let's discuss further on the run time in Q7.

6. [5 points]
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. [5 points]
Question. Include in your report a discussion on the running time

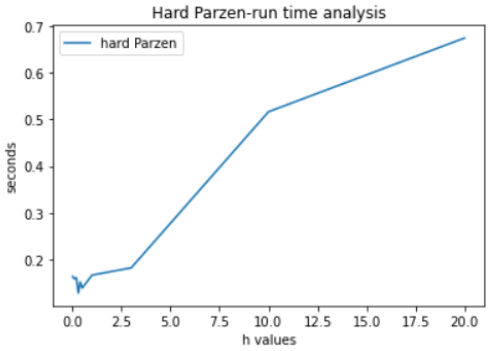
complexity of these two methods. How does it vary for each method when the hyperparameter h or σ changes? Why ?

ANS:

Let us consider, m is the dimensions(features/columns) in the training set and n is the number of training examples.

Hard Parzen Run time Complexity: O(m*n)

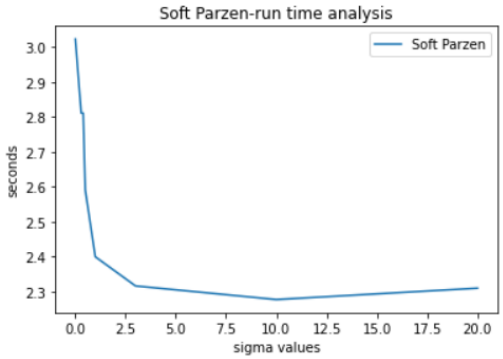
Hard Parzen suffers from the sparse-dimensionality curse i.e. as the dimensionality of the training set grows there would be larger volume in the hyperspace to capture the training points. The relationship is directly proportional, i.e. if the dimensionality grows the hyperspace grows. Hence, the distance between the points would increase and less points falls with the specified h i.e. less neighbours which might result in misclassification.



In terms of hyper-parameter(h): *As the h grows, the number of neighbours included with the region would be more, hence it results in some time to compute it. To summarize, as h grows the run time of the model grows.*

Soft Parzen Run time Complexity: O(m*n)

Soft Parzen suffers from dense-dimensionality curse. For every iteration on the test point, it is computed with every point in the training set and finding the Gaussian and multiplication makes soft Parzen computationally expensive (inner calculation of all the points) as it suffers from density. If the training points grows, the run time grows. When compared with Hard Parzen, the big O is same, however the based-on sparsity and density the run time varies between these two models.



In terms of hyper-parameter(Sigma): *As the sigma grows, there is no substantial increase in the run time as for every sigma value each test point is computed with every training point.*

8. [5 points]
Question. Implement a random projection (Gaussian sketch) map to be used on the input data:

Your function **random_projections** 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. $random\_projections(X_{n,4}, A_{4,2}) = X^{proj}_{n,2}$

9. [10 points]
Question. Similar to Question 5, compute the validation errors of Hard Parzen classifiers trained on 500 random projections of the training set, for

$$h \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0\}$$

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 500×10 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

$$\sigma \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0\}$$

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?

ANS:
When compared to the previous error rate analysis, its almost similar.

In terms of Bias Variance trade off, *When the hyper parameters values are less it couldn't able to capture the data points well, as a result there was high error rates when h or sigma is equal to 0.01 and the optimal value for, hard Parzen(h)= 1 and soft Parzen(sigma)= 0.3. After this point, the error rate is high due to variance in the model i.e. over-fitting.*

In terms of run time*By using random projections, we are reducing the dimension (from 4 to 2), it makes it better in terms of run time when compared to the previous result. However, in terms of Big O Notations it remains the same as O(m*n).*

To Summarize KNN algorithm,

- KNN suffers Bias and variance problem, hence choosing the right hyper-parameters would be essential for the model's generalized learning.
- KNN mainly depends on the distance metrics (like Euclidean/-Manhattan's). Hence, I personally think, it might suffer from outliers and need as good amount of pre-processing
- If we could reduce the dimensions through PCA or LDA or any other dimensionality approach, we could somewhat control the dimensionality curse in KNN algorithms for better performance.

- There is no as such training time for the KNN algorithms and the model doesn't learn any parameters and mainly depends of the hyper parameters. Hence it is good to have a good amount of training points with dimensions reduced, so that model can perform well good.