University of Pittsburgh
CS2750 Machine Learning
Professor Milos Hauskrecht

Handout 3
February 3, 2022

Problem assignment 3

Due: Thursday, February 10, 2022

Problem 1: Non-parametric density estimation

Non-parametric density estimation methods estimate a density with the help of all instances in the training data. In this problem we implement and study three basic non-parametric density methods.

- (a) You have received an initial file: hw-3-NDE.py you will need to gradually modify in order to answer the remaining parts of this problem. The initial code loads two data files. The first file is *true_pdf.csv* that gives you the 'true' pdf values for an ordered set of 500 one-dimensional x points. The points are in the first column, the pdf values are in the second column. The code then plots the pdf function based on the true pdf values. The second file that hw-3-NDE.py loads is the training data file NDE-data.csv

- (b) The non-parametric density estimation methods rely on observed instances in the training set. Write a function $Parzen\_window\_pdf(x, D, h)$ that estimates the probability density function at a data point x using the data in the training dataset D that fall into the region around x efined by the width parameter h. Include the function in file hw-3-NDE.py

- (c) Consider the training data loaded from $data\_NDE.csv$. Apply the $Parzen\_window$ function from Part (b) to all instances in the first column of *true_pdf.csv* file. Use instances in file $data\_NDE.csv$ as the training data D. Repeat this for three different values of parameter $h = 0.02, 0.05, 0.1$. For each value h, plot the estimate of the density based on the Parzen window together with the true density values in file *true_pdf.csv*. Include all three plots in the report. Analyze the results by comparing the true density values with the estimates, as well as, estimates based on the different values of h.

- (d) Now instead of the Parzen window assume we estimate the pdf for any instance x using a smooth Gaussian kernel. Write a function $Gaussian\_kernel\_pdf(x, D, h)$, that takes a data instance x and outputs its pdf based on the Gaussian kernel with the width parameters $h = 0.1, 0.16, 0.25$. Include it in file hw-3-NDE.py . Similarly to

Part (c), apply the pdf estimate based on the *Gaussian kernel* to all instances in the first column of *true pdf.csv* file. For each value of h, plot the estimate of the density based on the Gaussian kernel together with the true density values in file *true pdf.csv*. Include all three plots (for the different values of h) in the report. Analyze the results, and contrast them to results in Part c. item

(e) Finally, let us consider knn density estimators. Write and submit a function $[pdf_x] = knn\ estimate\ pdf(x, D, k)$ that returns a pdf estimate based on the k-nearest neighbor approach. Apply the pdf estimate based on the kNN to all instances in the first column of *true pdf.csv* file.In terms of k consider the following three values $k = 1, 3, 5$. For each k show the pdf estimate performance and compare it to the true density estimates. Analyze the results, and compare them to results obtained in Parts c and d.

In addition to your answers compiled in the report, please submit the final hw-3-NDE.py file with your code.

Problem 2. Linear regression

In this problem we will use the Boston Housing dataset from the CMU StatLib Library that concerns prices of housing in Boston suburbs. A data sample consists of 13 attribute values (indicating parameters like crime rate, accessibility to major highways etc.) and the median value of housing in thousands we would like to predict. The the description of the data is in the file *housing_desc.txt*.

Part 2.1. Linear model

Our goal is to predict the median value of housing based on the values of 13 attributes. You are given a file hw-3-lin-regression-part1.py that performs the following steps:

- Divides the data into: a training dataset and a testing dataset
- Fits the parameters of the linear model on the training data
- Applies the model to the testing data
- Reports the mean squared error of the model on the testing data

Please accomplish the following tasks by running and modifying the code in hw-3-lin-regression-part1.py

- (a) Run the code

- (b) Modify the code so that in addition to the testing error it also calculates the training error. Report the errors on both the training and testing set. Analyze the results. Which one is better? Do the results indicate we should be concerned about overfitting the model? Explain.

Submit the modified hw-3-lin-regression-part1.py file.

Part 2.2. Fitting a quadratic model

Make copy of the code in hw-3-lin-regression-part1.py and name it hw-3-lin-regression-part2.py file. Modify the code as follows:

- Write a function quadratic-expansion that takes the original input vector (with 13 attributes) and outputs a vector that consists of all linear and quadratic expansions of the original vector, such as $x_1, x_2, \cdots, x_d, x_1^2, x_2^2, \cdots, x_1x_2, x_1x_3, \cdots$. Please note the products such as x_1x_2 and x_2x_1 are the same and should be included only once !
- Report the dimension of the new vector after the expansion
- Use the quadratic-expansion function to build the new transformed input space X' from the original X. Learn the linear regression model for the newly transformed inputs. Calculate the mean squared errors. Compare the results to the linear model in Part 2.1. Which model is better? Should we be concerned about overfitting the model? Explain.

Part 2.3. Online (stochastic) gradient descent

The linear regression model can be also learned using the gradient descent method.

(a) Write and submit hw-3-lin-regression-part3.py that:

- Loads the housing dataset.
- Normalizes the X (input) part of the data fusing the standard normalization method. This procedure is implemented in the sklearn.preprocessing library and is named StandardScaler. The normalization is needed because the gradient descent method may become unstable when fed with un-normalized data. To prevent
- Splits the data to the training and testing set
- Implements the stochastic gradient descent procedure. It should:
  - start with all one weights (all weights set to 1 at the beginning);

  - update weights using the annealed learning rate $0.05/t$, where t denotes the t-th update step. Thus, for the first data point the learning rate is 0.05, for the second it is 0.025, for the 3-rd is 0.05/3 and so on;
  - repeat the update procedure for 1000 steps reusing the examples in the training data if necessary (hint: the index of the i-th example in the training set of size n can be obtained by (i mod n) operation);

- prints the final set of weights
- applies the model to both the training and testing data to computes and the mean squared errors for both the train and test data
- prints the mean train and test errors.

(b) Run your program and report the results in the report. Give the mean squared errors for both the training and test set. Is the result better or worse than the one obtained by solving the linear regression problem in Part 2.1.? Submit the in hw-3-lin-regression-part3.py

(c) Experiment with the gradient descent procedure. Try to use: fixed learning rate (say 0.05, 0.01), or the different number of update steps (say 500 and 3000). You may want to change the learning rate schedule as well. Try for example $2/\sqrt{n}$. Report your results and any interesting behaviors you observe.