University of Pittsburgh
CS1675 Introduction to Machine Learning — Handout 4
Professor Milos Hauskrecht — February 18, 2021

Problem assignment 4

Due: Thursday, February 25, 2021

Problem 1. 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 data are in the file *housing.txt*, the description of the data is in the file *housing_desc.txt*.

Part 3.1. Exploratory data analysis.

Examine the dataset housing.txt using Matlab. Answer the following questions.

- (a) How many binary attributes are in the data set? List the attributes.
- (b) Calculate and report correlations in between the first 13 attributes (columns) and the target attribute (column 14). What are the attribute names with the highest positive and negative correlations to the target attribute?
- (c) Note that the correlation is a linear measure of similarity. Examine scatter plots for attributes and the target attribute using the function you wrote in problem set 1. Which scatter plot looks the most linear, and which looks the most nonlinear? Plot these scatter plots and briefly (in 1-2 sentences) explain your choice.
- (d) Calculate all correlations between the 14 columns (using the corrcoef function). Which two attributes have the largest mutual correlation in the dataset?

Part 3.2. Linear regression

Our goal is to predict the median value of housing based on the values of 13 attributes. For your convenience the data has been divided into two datasets: (1) a training dataset *housing_train.txt* you should use in the learning phase, and (2) a testing dataset *housing_test.txt* to be used for testing.

Assume that we choose a linear regression model to predict the target attribute. Using Matlab:

- (a) Write and submit a function *LR_solve*(X, y) that takes **X** and **y** components of the training data (**X** is a matrix of inputs where rows correspond to examples) and returns a vector of weights **w** with the minimal mean square fit. (Hint: you can use backslash operator '/' to do the least squares regression directly; check Matlab's help).
- (b) Write a function *LR_predict* that takes input components of the test data (**X**) and a fixed set of weights (**w**), and computes vector of linear predictions **y**.
- (c) Write (and submit) a matlab program (script) *main_LR.m* that:
  - loads the train and test set,
  - learns the weights of the linear regression model from the training data (use function from part a to do so)
  - prints the weights of the model
  - applies the model to both the training and testing data (use the function from part b), and computes the mean squared error on the training and testing data set
  - prints the training and testing errors
- (d) In your report please list the resulting weights, and both mean square errors. Compare the errors for the training and testing set. Which one is better? Does the result make sense? Explain.

Part 3.3. Online (stochastic) gradient descent

The linear regression model can be also learned using the gradient descent method. One concern when using the gradient descent method is that it may become unstable when fed with un-normalized data. To deal with the issue you are given two matlab functions: *compute_norm_parameters* and *normalize* that are able to respectively calculate the normalization coefficients from the data matrix and apply them to un-normalized data matrix.

(a) Write and submit matlab program (script) *main_LR_SGD.m* that:

- loads the train and test set
- Normalizes the x (input) part of both the train and test data. This is done by first calculating the means and stds of all input attributes on the train data and then applying these to normalize both the train and test inputs. Use functions *compute_norm_parameters* and *normalize* functions given to you for this purpose.

- 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.

Run your program and report the results in the report. Give the mean errors for both the training and test set. Is the result better or worse than the one obtained by solving the linear regression problem exactly.

(b) Create *main_LR_SGD_v2.m* by copying and modifying *main_LR_SGD.m* program from part a. (please keep the original copy) without the data normalization step, that is, run it on the original un-normalized data. Please report what you have observed?

(c) Create *main_LR_SGD_v3.m* by copying and modifying *main_LR_SGD.m* from part a (please keep the original copy for submission !) such that it lets you to progressively observe changes in the mean train and test errors during the execution of the gradient descend procedure. Use functions *init_progress_graph* and *add_to_progress_graph* given to you. The *init_progress_graph* initializes the graph structure and *add_to_progress_graph* lets you add new data entries on-fly to the graph. Using the two functions plot the mean squared errors for the training and test test after every 50 iteration steps. Submit the program and include the graph in the report.

(d) Experiment with the gradient descent procedure. Try to use: fixed learning rate (say 0.05, 0.01), or 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.