Linear Regression

In this problem set we 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*.

Problem 1.1. 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 by writing your own function. 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?

Problem 1.2. 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 a function *LR_solve* that takes **X** and **y** components of the data (**X** is a matrix of inputs where rows correspond to examples) and returns a vector of coefficients **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 the program *main3_2.m* that loads the train and test set, learns the weights for the training set, and computes the mean squared error of your predictor on both the training and testing data set.

• (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?

Problem 1.3. Online gradient descent

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

(a) Implement an online gradient descent procedure for finding the regression coefficients w. Your program should:

• start with zero weights (all weights set to 0 at the beginning);

• update weights using the annealed learning rate 2/*t*, where t denotes the *t* -th update step. Thus, for the first data point the learning rate is 2, for the second it is 2/2 = 1, for the 3-rd is 2/3 and so on;

• repeat the update procedure for 1000 steps reusing the examples in the training data if neccessary (hint: the index of the i-th example in the training set can be obtained by (i mod n) operation);

• return the final set of weights.

(b) Write a program *main3_3.m* that runs the gradient procedure on the data and at the end prints the mean test and train errors. **Your program should normalize the data before running the method**. Run it and report the results. Give the mean errors for both the training and test set. Is the result better or worse than the one obtained by solving the regression problem exactly?

(c) Run the gradient descent on un-normalized dataset. What happened?

(d) Modify *main3_3.m* from part b, such that it lets you to progressively observe changes in the mean train and test errors. Use functions *init_progress_graph* and *add_to_progress_graph* provided. 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 set for 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 many want to change the learning rate schedule as well. Try for example $2/\sqrt{n}$. Report your results and any interesting behaviors you observe.

Problem 1.4. Regression with polynomials.

Assume we are not happy with the predictive accuracy of the linear model and we decide to explore a more complex model for predicting housing values. Assume we decide to use a quadratic polynomial to model the relation between *y* and x:

$$f(\mathrm{x},\mathrm{w}) = w_0 + \sum_{i=1}^{13} w_i x_i + \sum_{i=1}^{13} \sum_{j=1}^{13} w_{ij} x_i x_j$$

• (a) Write a function *extendx* that takes an input **x** and returns an expanded **x** that includes all linear and degree two polynomials.

• (b) What happened to the binary attribute after the transformation?

• (c) Write and submit a Matlab program *main3_4.m* that computes the regression coefficients for the extended input and both train and test errors for the result.

• (d) Report both errors in your report and compare them with the results in part 2. What do you see? Which method would you use for the prediction? Why? Please do not turn in the weights for this part in your report.

Classification

Problem 2.1. Data analysis

The dataset we use in this problem set is very simple and consists of a two-dimensional input and a class label. The data are available on the course web page and are divided into two files - one used for training (*classification_train.txt*), the other for testing (*classification_test.txt*). The data in files are in rows such that first two columns represent inputs and the last (third) column the class label (0 or 1).

Since inputs are only two dimensional we can easily visualize the data for the two classes in a 2D plot.

• Write a program that plots the input data points in *classification_train.txt* such that the plot distinguishes between data points with different class labels (use different color and symbol for a point, e.g. ’x’ or ’o’).

• Include the plot in your report. Is it possible to separate the two classes perfectly with a linear decision boundary?

Problem 2.2. Logistic regression

We are interested in building a classifier based on the logistic regression model and the gradient optimization methods.

• (a) During the class you were given the expression for the gradient of the logistic regression model. Use the loglikelihood setup from the lecture to derive the expression.

Show clearly the steps of the derivation. Please remember that the ’default’ gradient takes into account all datapoints in the training set.

• (b) Write and submit a gradient procedure *GLR.m* for updating the parameters of the logistic regression model. Your gradient procedure should:

– start from unit weights (all weights set to 1 at the beginning);
– use the annealed learning rate $2/k$;
– executes for K steps where K is the parameter of the procedures.

• (c) Write and submit a program *main2.m* that runs the GLR function for 500 steps and after the training computes mean misclassification errors for both the training and test set. In your report include, the resulting weights, and misclassification errors.

• (d) Update the *main2.m* with plot functions that let you observe the progress of the errors after every 50 update steps. Use functions defined earlier for this purpose. Include the resulting graph in your report.

• (e) Experiment with the GLR function by: (I) changing the number of steps K and (II) trying different learning rates. In particular, try some constant learning rates and $1/\sqrt{k}$ learning rate schedule. Report the results and graph from your experiments and explanations of behaviors you have observed.

Problem 2.3. Generative classification model

An alernative approach is to learn a generative model with class-conditional densities and use the parameters of such a model to do the prediction.

Assume that an input x for each class (c = 0 or 1) follows a multivariate normal distribution. That is,

$$p(\mathrm{x} \mid c = 0) \sim N(\mu_0, \Sigma_0)$$
$$p(\mathrm{x} \mid c = 1) \sim N(\mu_1, \Sigma_1)$$

Further assume that the prior probability of a class is represented by a Bernoulli distribution.

Parameters of the generative model can be computed from the training data using density estimation techniques, such as maximum likelihood estimation. Once this is ccomplished, we can use the estimates to make class predictions for new inputs.

Let $\tilde{\Theta} = \{\tilde{\mu}_0, \tilde{\Sigma}_0, \tilde{\mu}_1, \tilde{\Sigma}_1, \tilde{\theta}_{c=1}\}$ represent parameter estimates. To predict the class we use discriminant functions based on the posterior probability of a class given the input and model parameters. This can be computed via Bayes rule:

$$g_1(\mathrm{x}) = p(c=1 \mid \mathrm{x}, \tilde{\Theta}) = \frac{p(\mathrm{x} \mid \tilde{\mu}_1, \tilde{\Sigma}_1)\,\tilde{\theta}_{c=1}}{p(\mathrm{x} \mid \tilde{\mu}_0, \tilde{\Sigma}_0)(1-\tilde{\theta}_{c=1}) + p(\mathrm{x} \mid \tilde{\mu}_1, \tilde{\Sigma}_1)\,\tilde{\theta}_{c=1}}$$

Assume we want to use a generative model in which the two class conditional densities share the same covariance matrix Σ. That is:

$$p(\mathrm{x} \mid c=0) \sim N(\mu_0, \Sigma)$$
$$p(\mathrm{x} \mid c=1) \sim N(\mu_1, \Sigma)$$

Provide the following answers:

• (a) Give the formula for computing ML estimates of means of class conditional densities?

• (b) How would you go about computing the estimate of the covariance matrix Σ? Note that the estimate of Σ must combine both class 0 and class 1 examples.

• (c) How would you estimate the prior of class $\tilde{\theta}_{c=1}$ =1?

• (d) Implement function *Max_Likelihood* that computes the estimates of model parameters using the training set.

• (e) Implement the function *Predict_class* that chooses the class using the discriminant functions based on class posteriors.

• (f) Write and submit a program *main4.m* that learns the generative model and then uses it to compute the predictions. The program should compute mean misclassification errors for both training and testing datasets.

• (g) Report the results (parameters of the generative model), and errors. Compare them to the results obtained in problem 2.