CSCI 567, Fall 2018
Haipeng Luo

Homework #1 Programming Assignments

Due: 11:59 pm, Sept 16, 2018

Instructions

Submission: Assignment submission will be via your Github repo (remember to "push"; "commit" is not enough). Please be sure to follow all directions outlined here.

You can submit multiple times, but only *the last submission* counts. That means if you finish some problems and want to submit something first and update later when you finish, that's fine. In fact you are encouraged to do this: that way, if you forget to finish the homework on time or something happens (remember Murphy's Law), you still get credit for whatever you have turned in.

In the following, we will first explain the general setup, the coding pipeline you will use in Sect. 1 and 2, and the datasets.

Setup This assignment is to be done in python3 (version: 3.5.2). Make sure you have the correct version installed. You will also need following python packages :

- numpy (1.13.1)
- scipy (0.19.1)
- matplotlib (2.0.2)
- pandas (0.20.3)

There are multiple ways you can install python3, for example:

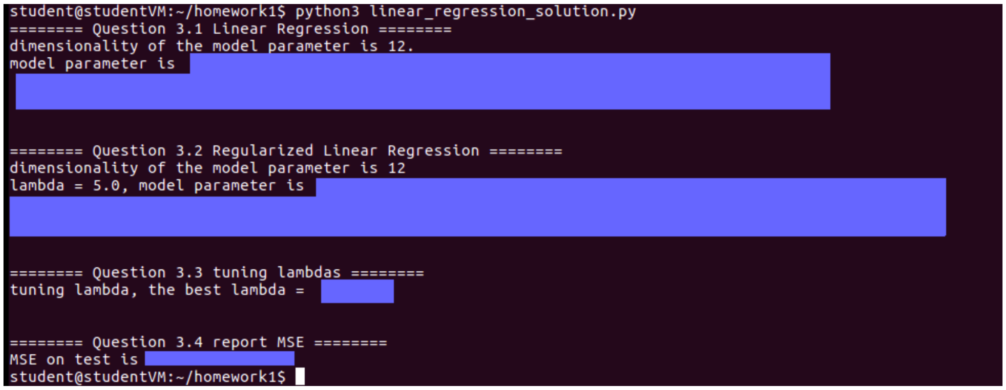
- [Recommended] Install this virtual machine (VM) in virtualbox. Everything is setup already in this VM. You submission will eventually be graded within this VM too.
- Alternatively you can also use virtualenv to configure a python3 environment for all programming assignments.

Pipeline A standard machine learning pipeline usually consists of three parts. 1) Load and pre-process the data. 2) Train a model on the training set and use the validation set to tune hyperparameters. 3) Test the final model and report the result. In this assignment, we will provide you a template for each section (`linear_regression.py, knn.py`), which follows this 3-step pipeline. We provide the data loading and preprocessing codes in step 1 and define the output format in step 3. You will be asked to implement step 2's algorithms to complete the pipeline. Do not make any modification to our implementations for step 1 and 3.

Please do not import packages that are not listed in the provided scripts. Follow the instructions in each section strictly to code up your solutions. *DO NOT CHANGE THE OUTPUT FORMAT. DO NOT MODIFY THE CODE UNLESS WE INSTRUCT YOU TO DO SO.* A homework solution that mismatches the provided setup, such as format, name, initializations, etc., *will not* be graded. It is your responsibility to make sure that your code runs with python3 on the VM.

Datasets

Regression Dataset The UCI Wine Quality dataset lists 11 chemical measurements of 4898 white wine samples as well as an overall quality per sample, as determined by wine connoisseurs. See **winequality-white.csv**. We split the data into training, validation and test sets in the preprocessing code. You will use linear regression to predict wine quality from the chemical measurement features.



```
student@studentVM:~/homework1$ python3 linear_regression_solution.py
======== Question 3.1 Linear Regression ========
dimensionality of the model parameter is 12.
model parameter is [redacted]


======== Question 3.2 Regularized Linear Regression ========
dimensionality of the model parameter is 12
lambda = 5.0, model parameter is [redacted]


======== Question 3.3 tuning lambdas ========
tuning lambda, the best lambda = [redacted]


======== Question 3.4 report MSE ========
MSE on test is [redacted]
student@studentVM:~/homework1$
```

Figure 1: Example output

Classification Dataset MNIST is one of the most well-known datasets in computer vision, consisting of images of handwritten digits from 0 to 9. We will be working with a subset of the official version of MNIST, denoted as **mnist_subset**. In particular, we randomly sampled 700 images from each category and split them into training, validation, and test sets. This subset corresponds to a JSON file named **mnist_subset.json**. JSON is a lightweight data-interchange format, similar to a dictionary. After loading the file, you can access its training, validation, and test splits using the keys 'train', 'valid', and 'test', respectively. For example, if we load **mnist_subset.json** to the variable x, $x['train']$ refers to the training set of **mnist_subset**. This set is a list with two elements: $x['train'][0]$ containing the features of size N (samples) $\times D$ (dimension of features), and $x['train'][1]$ containing the corresponding labels of size N.

Example output For `linear_regression.py` in Problem 1, you should be able to run it on VM and see output similar to Fig. 1.

Collaboration: Please consult the syllabus for what is and is not acceptable collaboration. Review the rules on academic conduct in the syllabus: a single instance of plagiarism can adversely affect you significantly more than you could stand to gain.

Problem 1 Linear Regression (30 points)

You are asked to implement 4 python functions for linear regression. The input and output of the functions are specified in `linear_regression.py`. You should be able to run `linear_regression.py` after you finish the implementation. Note that we have already appended the column of 1's to the feature matrix, so that you do not need to modify the data yourself.

Problem 1.1 Linear regression (6 points)
Implement linear regression and return the model parameters. *What to submit:* fill in the function `linear_regression_noreg(X, y)`.

Problem 1.2 Regularized linear regression (10 points)
To prevent overfitting, we usually add regularization. For now, we will focus on L_2 regularization. In this case, the optimization problem is:

$$\boldsymbol{w}^{\lambda} = \arg\min_{\boldsymbol{w}} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{w}||_2^2 \tag{1}$$

where $\lambda \geq 0$ is a hyper-parameter used to control the complexity of the resulting model. When $\lambda = 0$, the model reduces to the usual (unregularized) linear regression. For $\lambda > 0$ the objective function balances between two terms: (1) the data-dependent quadratic loss function $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$, and (2) a function of the model parameters $||\boldsymbol{w}||_2^2$.

Implement your regularized linear regression algorithm.
What to submit: fill in function `regularized_linear_regression(X, y, λ)`.

Problem 1.3 Tuning the regularization hyper-parameter (9 points)
Use the validation set to tune the regularization parameter $\lambda \in \{0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$. We select the best one that results in the lowest mean square error on the validation set.
What to submit: fill in the function `tune_lambda(Xtrain, ytrain, Xval, yval, lambds)`.

Problem 1.4 Mean square error (5 points)
Report the mean square error of the model on the give n test set.
What to submit: fill in the function `test_error(w, X, y)`.

Problem 2 *k* Nearest Neighbors

(20 points)

Review In the lecture, we define the classification rule of the k-nearest neighbors (kNN) algorithm for an input example x as

$$v_c = \sum_{x_i \in \text{knn}(x)} \mathbb{1}(y_i == c), \forall c \in [C] \tag{2}$$

$$y = \arg\max_{c \in [C]} v_c \tag{3}$$

where $[C]$ is the set of classes.

A common distance measure between two samples x_i and x_j is the Euclidean distance:

$$d(x_i, x_j) = \|x_i - x_j\|_2 = \sqrt{\sum_d (x_{id} - d_{jd})^2}. \tag{4}$$

You are asked to implement 4 python functions for kNN. The input and output are specified in `knn.py`. You should be able to run `knn.py` after you finish the implementation.

Problem 2.1 Distance calculation **(3 points)**
Compute the distance between test data points in $\boldsymbol{X}$ and training data points in $\boldsymbol{X}_{\text{train}}$ based on Eqn. 4.
What to submit: fill in the function `compute_distances(Xtrain, X)`.

Problem 2.2 kNN classifier **(5 points)**
Implement kNN classifier based on Eqn. 3. Your algorithm should output the predictions for the test set. *Important:* You do not need to worry about ties in distance when finding the k nearest neighbor set. However, when there are ties in the majority label of the k nearest neighbor set, you should return the label with the smallest index. For example, when $k = 5$, if the labels of the 5 nearest neighbors happen to be $1, 1, 2, 2, 7$, your prediction should be the digit 1.
What to submit: fill in the function `predict_labels(k, ytrain, dists)`.

Problem 2.3 Report the accuracy **(6 points)**
The classification accuracy is defined as:

$$\text{accuracy} = \frac{\text{\# of correctly classified test examples}}{\text{\# of test examples}} \tag{5}$$

(The accuracy value should be in the range of $[0, 1]$)
What to submit: fill in the code for function `compute_accuracy(y, ypred)`.

Problem 2.4 Tuning k **(6 points)**
Find k among $[1, 3, 5, 7, 9]$ that gives the best classification accuracy on the validation set.
What to submit: fill in the code for function `find_best_k(K, ytrain, dists, yval)`.