

Homework 2

Due Friday April 19, 2019, by 11:59pm

Instructions: All coding exercises must be completed in Python. Upload your answers to the questions below to Canvas. Submit the answers to the questions in a PDF file and your code in a (single) separate file. Be sure to comment your code to indicate which lines of your code correspond to which question part. There is 1 reading assignment and 4 exercises in this homework.

Reading Assignment

Read Sec. 4.1 to 4.4.2 and Sec. 7.10 in *The Elements of Statistical Learning*.

1 Exercise 1

In this exercise, you will implement a first version of *your own gradient descent algorithm* to solve the ridge regression problem. Throughout the homeworks, you will keep improving and extending your gradient descent optimization algorithm. In this homework, you will implement a basic version of the algorithm.

Recall from Week 1 and Week 2 Lectures that the ridge regression problem writes as

$$\min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (y_i - x_i^T \beta)^2 + \lambda \|\beta\|_2^2, \quad (1)$$

that is, if you expand,

$$\min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \left(y_i - \sum_{j=1}^d \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^d \beta_j^2. \quad (2)$$

1.1 Remarks

Several remarks are in order.

Normalization Note that there is a $1/n$ normalization factor in the empirical risk term in the equations. Note also that there is a λ multiplicative factor in the regularization penalty term in the equations. Sometimes, in articles, you may see the normalization $\lambda/2$ instead for the ℓ_2^2 -regularization penalty. This is convenient when you compute the gradient of that term because the 2 and the $1/2$ cancel.

You can actually normalize the terms any way you want *as long as you are consistent all the way through* in your mathematical derivations, your codes, and your experiments, especially when you do cross-validation.

So here is my general advice:

- do normalize the empirical risk term so that it is an average, not a sum; this normalization will be important for large scale problems where the sum can become very large.
- check what optimization problem exactly is solved when you use a library, so you can compare your solution to the optimization problem to the solution found by the library and compare the optimal value of the regularization found by your cross-validation to the one found the library's cross-validation.

Intercept It is common in traditional statistics and machine learning books and libraries to include an intercept β_0 in the statistical model. Having a separate intercept coefficient is actually not that important, and provably so, especially if the data was properly centered and standardized beforehand.

There is actually a simple way to bypass the issue of having a separate intercept coefficient by adding a constant variable 1 in the variables. See Sec. 2.3.1 of *The Elements of Statistical Learning*. So the d variables in the equations correspond to the $(d - 1)$ original variables plus 1 dummy variable equal to 1.

1.2 Gradient descent

The gradient descent algorithm is an iterative algorithm that is able to solve differentiable optimization problems such as (1). Define

$$F(\beta) = \frac{1}{n} \sum_{i=1}^n (y_i - x_i^T \beta)^2 + \lambda \|\beta\|_2^2. \quad (3)$$

Gradient descent generates a sequence of iterates¹ (β_t) that converges to the optimal solution β^* of (1). The optimal solution of (1) is defined as

$$F(\beta^*) = \min_{\beta \in \mathbb{R}^d} F(\beta). \quad (4)$$

Gradient descent is outlined in Algorithm 1. The algorithm requires a sub-routine that computes the gradient for any β . The algorithm also takes as input the value of the constant step-size η .

- Assume that $d = 1$ and $n = 1$. The sample is then of size 1 and boils down to just (x, y) . The function F writes simply as

$$F(\beta) = (y - x\beta)^2 + \lambda\beta^2. \quad (5)$$

Compute and write down the gradient ∇F of F .

¹The subscript t refers to the iteration counter here, not to the coordinates of the vector β .

- Assume now that $d > 1$ and $n > 1$. Using the previous result and the linearity of differentiation, compute and write down the gradient $\nabla F(\beta)$ of F .
- Consider the **Hitters** dataset, which you should load and divide into training and test sets using the code below.²

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

# Load the data
hitters = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/'
                     'master/Hitters.csv', sep=',', header=0)
hitters = hitters.dropna()

# Create our X matrix with the predictors and y vector with the response
X = hitters.drop('Salary', axis=1)
X = pd.get_dummies(X, drop_first=True)
y = hitters.Salary

# Divide the data into training and test sets. By default, 25% goes into the test set.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

Standardize the data. Note that you can convert a data frame into an array by using `np.array()`.

- Write a function *computegrad* that computes and returns $\nabla F(\beta)$ for any β . Avoid using `for` loops by vectorizing the computation.
- Write a function *graddescent* that implements the gradient descent algorithm described in Algorithm 1. The function *graddescent* calls the function *computegrad* as a sub-routine. The function takes as input the initial point, the constant step-size value, and the maximum number of iterations. The stopping criterion is the maximum number of iterations.
- Set the constant step-size to $\eta = 0.05$ and the maximum number of iterations to 1000. Run *graddescent* on the training set of the **Hitters** dataset for $\lambda = 0.05$. Plot the curve of the objective value $F(\beta_t)$ versus the iteration counter t . (Again, avoid using `for` loops when computing the objective values). What do you observe?
- Denote β_T the final iterate of your gradient descent algorithm. Compare β_T to the β^* found by *sklearn.linear_model.Ridge*. Compare the objective value for β_T to the one for β^* . What do you observe?
- Run your gradient algorithm for many values of η on a logarithmic scale. Find the final iterate, across all runs for all the values of η , that achieves the smallest value of the objective. Compare β_T to the β^* found by *sklearn.linear_model.Ridge*. Compare the objective value for β_T to the β^* . What conclusion do you draw?

²You may encounter problems with the quotes when copying and pasting it. If so, delete the quotes that are there and retype the quotes.

Algorithm 1 Gradient Descent algorithm with fixed constant step-size

input step-size η
initialization $\beta_0 = 0$
repeat for $t = 0, 1, 2, \dots$
 $\beta_{t+1} = \beta_t - \eta \nabla F(\beta_t)$
until the stopping criterion is satisfied.

2 Exercise 2

Exercise 3.8 in Chapter 3 of *An Introduction to Statistical Learning* (in Python):

This question involves the use of simple linear regression on the **Auto** data set.

- (a) Read in the dataset. The data can be downloaded from this url: <http://www-bcf.usc.edu/~gareth/ISL/Auto.csv> When reading in the data use the option `na_values='?'`. Then drop all NaN values using `dropna()`.
- (b) Use the `OLS` function from the `statsmodels` package to perform a simple linear regression with `mpg` as the response and `weight` as the predictor. Be sure to include an intercept. Use the `summary()` attribute to print the results. Comment on the output. For example:

- (i) Is there a relationship between the predictor and the response?
- (ii) How strong is the relationship between the predictor and the response?
- (iii) Is the relationship between the predictor and the response positive or negative?

Hint: See this URL for help with the `statsmodels` functions: <http://www.statsmodels.org/dev/regression.html#examples>

- (c) Plot the response and the predictor using the `plot_fit` function (http://www.statsmodels.org/dev/generated/statsmodels.graphics.regressionplots.plot_fit.html)
- (d) Plot the residuals vs. fitted values. Comment on any problems you see with the fit.

3 Exercise 3

Exercise 3.9 in Chapter 3 of *An Introduction to Statistical Learning* (in Python):

This question involves the use of multiple linear regression on the **Auto** data set.

- (a) Produce a scatterplot matrix which includes all of the variables in the data set using `pandas.plotting.scatter_matrix`.
- (b) Compute the matrix of correlations between the variables using the `corr()` attribute in Pandas.

- (c) Use the `OLS` function from the `statsmodels` package to perform a multiple linear regression with `mpg` as the response and all other variables except `name` as the predictors. Be sure to include an intercept. Print the results. Comment on the output. For instance:
 - (i) Is there a relationship between the predictors and the response?
 - (ii) Which predictors appear to have a statistically significant relationship to the response?
 - (iii) What does the coefficient for the `year` variable suggest?
- (d) Plot the residuals vs. fitted values. Comment on any problems you see with the fit.
- (e) Statsmodels allows you to fit models using R-style formulas. See http://www.statsmodels.org/dev/example_formulas.html. Use the `*` and `:` symbols to fit linear regression models with interaction effects. Do any interactions appear to be statistically significant?
- (f) Try a few different transformations of the variables, such as $\log(X)$, \sqrt{X} , X^2 . Comment on your findings.

4 Exercise 4

Exercise 3.12 in Chapter 3 of *An Introduction to Statistical Learning* (in Python):
 This problem involves simple linear regression without an intercept.

- (a) Recall that the coefficient estimate β for the linear regression of Y onto X without an intercept is given by (3.38). Under what circumstance is the coefficient estimate for the regression of X onto Y the same as the coefficient estimate for the regression of Y onto X ?
- (b) Generate an example in Python with $n = 50$ observations in which the coefficient estimate for the regression of X onto Y is different from the coefficient estimate for the regression of Y onto X .
- (c) Generate an example in Python with $n = 50$ observations in which the coefficient estimate for the regression of X onto Y is the same as the coefficient estimate for the regression of Y onto X .