2 Regression

Course: COMPSCI 589 Machine Learning, Spring, 2021

Instructor: Justin Domke

Assignment: 2

Group work policy: You are allowed to complete this homework in teams of at most 3 students. You can use the Group Finder to help you find other students to work with. However, each student must submit their own individual .pdf and .zip files to Gradescope. List your team members at the beginning of your `report.pdf`. You may verbally discuss the assignment with course staff or students outside your group. Please also list any students or course staff (separately) at the beginning of your `report.pdf`. However, you may not *look, copy, or show* any part of another student's assignment. Copying any part of another assignment — even a single sentence or line of code — from anyone outside your team is considered plagiarism. We use sophisticated tools to detect this. Please do not do it.

Due date: March 3, 5:00 pm

Submission instructions:

- For this assignment, you should prepare your solutions in one of three formats:
  - Latex (any style)
  - Markdown
  - Jupyter notebook
- Regardless of how you prepared the solutions, you should export a single .pdf file that you upload to Gradescope. The .pdf should be submitted to the Assignment 2: Regression assignment. Most coding question will ask you to include your code as text in the solution .pdf.

- Additionally, you **must submit a .zip file** to Assignment 2: ZIP file in Gradescope. Your .zip file should contain four things:
    - `report.pdf` - Your report. 2
    - `report_src/` - A directory containing all source files for the report. 4
    - `code/` - A directory containing Python code for all parts of the assignment. 2
    - `code/run_me.py` - A single Python file that will generate all figures included in your report.

- If you use a Jupyter notebook, nothing changes. You still must put your code in external files, and you still must submit both a single .pdf and a .zip file containing the above components to the respective assignments in Gradescope.
    - Again, if you use a Jupyter notebook to create your pdf, **nothing changes**. All the requirements are exactly the same as if you used latex or markdown.
- When you submit the .pdf to Gradescope, you must mark page numbers for the different questions. We hate to do it, but we will penalize anyone who does not do this, as it creates a huge amount of difficulty for the graders.
- For the purpose of late days, the later of your two submissions will be considered the submission time for your assignment. E.g., if you submit your .pdf on time, but the .zip is two days late, the assignment will be considered two days late.
- Each question below is worth 10 points.

Building your own regression methods

In this first part of the assignment, you will build and run your own simple versions of regression methods. For these questions you may not use any package other than `numpy`. (In particular, you may not use `sklearn.`)

data.npz 35.3KB

You are given a file `data.npz` with training data consisting of 100 examples in 3 dimensions and test data consisting of 1000 examples in 3 dimensions. You can access the data as

```
stuff=np.load("data.npz") X_trn = stuff["X_trn"] y_trn =
stuff["y_trn"] X_val = stuff["X_val"] y_val = stuff["y_val"]
```

Question 1: Write a method to do K-nearest neighbors regression. Your method should have the following signature

```
def KNN_reg_predict(X_trn, y_trn, x, K): # do stuff here return y
```

where `X_trn` is a 2D array of training inputs, `y_trn` is a 1D array of training outputs, `x` is a 1D array of a single input to make predictions for, and `K` is the number of neighbors. The return value `y` is just a scalar.

Include your code as text in your solution file. You may break ties arbitrarily.

Question 2: For each value of `K` between 1 and 10 use your code from the previous question to mean training error and test error, evaluated using the squared loss. Give your answer as a 10x2 table, with one row for each value of `K` and one column for each of training / test error.

Now, repeat the process, except using absolute error instead of squared error.

Question 3: Write a method to evaluate a linear regression model. Your method should have the signature

```
def linear_reg_predict(x, w): # do stuff here return y
```

where `x` is a 1D array with a single input, and `w` is a 1D array of regression coefficients of the same length. The return value `y` is just a scalar.

Include your method as text in your solution file.

Question 4: Write a method to train a ~~linear~~ ridge regression model. Your method should have the signature

```
def linear_reg_train(X_trn, y_trn, λ): # do stuff here return w
```

where `X_trn` is a 2D array of training inputs, `y_trn` is a 1D array of training outputs, and `λ` is the regularization constant. (Use `l` instead of `λ` if you don't like Greek letters in your code.) The return value `w` is a 1D array of regression coefficients of the same size as `X_trn.shape[1]`. You should return the value `w` such that 6

$$\sum_{n=1}^{N}(w^\top x^{(n)} - y^{(n)})^2 + \lambda\|w\|^2$$

2

is minimized, where $x^{(n)}$ represents the different rows of `X_trn` and $y^{(n)}$ represents the entries of `y_trn`. Include your method as text in your solution file.

Note: In these questions, you do not need to add a "bias term". Just fit a linear model of the form $f_w(x) = w^\top x$.

Question 5: Repeat the evaluation process from question 2, except with different values of `λ` on the rows. Use the values λ=0, 0.001, 0.01, 0.1, 1, and 10.

Question 6: The errors you get in the previous question depend on the particular dataset you used. Consider each of the following errors:

1. Squared training error.
2. Squared test error.
3. Absolute training error.
4. Absolute test error.

For each of the errors above, which of the following describes how it changes as `λ` increases for *all* datasets? Pick one of (A)-(E), and explain why in at most 2 sentences: 3

A) Error only increases (or stays constant).

B) Error only decreases (or stays constant).

C) Error goes down for a while, then goes up. (It could stay constant in either phase.)

D) Error goes up for a while, then goes down. (It could stay constant in either phase.)

E) None of the above are guaranteed to happen.

Question 7: Write a method to evaluate a regression stump. Your method should have the signature

```
def reg_stump_predict(x, dim, thresh, c_left, c_right): # do stuff
here return y
```

Your function should return `c_left` if `x[dim]<=thresh` and `c_right` otherwise.

Question 8: Write a method to train a regression stump. Your method should have the signature

```
def reg_stump_train(X_trn, y_trn): # do stuff here return dim,
thresh, c_left, c_right
```

where `X_trn` is a 2D array of training inputs, and `y_trn` is a 1D array of training outputs. The values `dim`, `thresh`, `c_left`, and `c_right` are as in the prediction function above. You should return the values such that

$$\sum_{n=1}^{N}(f(x^{(n)}) - y^{(n)})^2$$

is minimized, where f represents your function from the previous question.

Include your method as text in your solution file.

Question 9: For the same data as above, train and evaluate a regression stump using your code from the previous two questions. Give a table with the training squared error, the test squared error, the training absolute error, and the test absolute error.

Note (Added Mar. 2): For each of these, we intended for you to report the *mean* error over each dataset. However, since this was ambiguous up to this point, we will also give full credit if you report the sum of the errors. Please do use the mean if you see this note as you'll get more insight. But no need to go to extra effort to redo your work if you've already completed it.

Running methods on real data

big_data.npz 897.0KB

For the following questions, you will use a larger dataset, given as `big_data.npz`. This file contains 4096 training and 4096 validation examples. There are 13 dimensions. (The first of these is 1.0, so you do not need to add a bias term yourself.)

For each of the following methods, compute the mean training and test error. In all cases, use squared error. You may use methods from `sklearn`, or you may implement these methods yourself.

Question 10: K-nearest neighbors regression, with the following values of `K`: 1, 2, 5, 10, 20, 50.

Question 11: Regression trees trained to (greedily) minimize squared error. Grow regression trees to the following maximum depths: 1,2,3,4,5. 3

Question 12: Ridge regression with each of the following regularization constants λ: 0, 1, 10, 100, 1000, 10000.

Note: You should *train* to minimize the *sum* of error with the regularization penalty, i.e. $\sum_{n=1}^{N}(w^\top x^{(n)} - y^n)^2 + \lambda ||w||^2$. However, you should still *report mean* errors, without any regularization penalty included, just like with any other method.

Question 13: Lasso regression with each of the following regularization constants λ: 0, .1, 1, 10, 100, 1000.

Again, train to minimize the sum of the error and the regression penalty, i.e. $\sum_{n=1}^{N}(w^\top x^{(n)} - y^{(n)})^2 + \lambda ||w||_1$, but report mean errors without any regularization constant included.

Hint: The Lasso method in `sklearn` minimizes an expression that we would write in our notation as $\frac{1}{2N}\sum_{n=1}^{N}(w^\top x^{(n)} - y^{(n)})^2 + \alpha\|w\|_1$. You must choose α carefully so that solving this is equivalent to solving the stated problem with a given λ.

Group Finder

FAQ