CMPSCI 390A Homework 3

YOUR NAME HERE

Assigned: Feb 11 2018; Due: Feb 18 2018 @ 11:00 am ET

Abstract

This assignment is the second part of a two-part assignment on linear regression. In this assignment, you will create a program that uses gradient descent to approximate a linear model's optimal parameters for a particular data set. You will then compare the linear model to a k-NN implementation that we provide, describing how well each method can fit the data and how easy it was to use each method. To submit this assignment, upload a `.pdf` to Gradescope containing your responses to the written response questions below. You are required to use LaTeX for your write up. When submitting your answers, use the template LaTeX code provided and put your answers below the question they are answering. Do not forget to put your name on the top of the .pdf. To submit the assignment's coding portion, upload a single python file called `my_hw3.py`, to the Gradescope programming assignment for Homework 3. An auto-grader will check your code for correct output. As such, your program must meet the requirements specified below. We will also be using cheating detection software, so, as a reminder, you are allowed to discuss the homework with other students, but you must write the code on your own.

1 Gradient Descent for Linear Regression (60 points)

As a follow up to the previous homework, in this assignment, you will implement a gradient descent algorithm for optimizing the weights of a linear model. In this assignment we will use the *least squares* loss function from class, but with an extra $\frac{1}{2}$ scaling constant:

$$l(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_w(x_i))^2.$$

Recall that the gradient descent procedure minimizes the loss function $l(w)$ by moving the weight vector w in the opposite direction of the gradient (vector of partial derivatives) of the loss function, i.e.,

$$\begin{aligned} w_{k+1} &\leftarrow w_k - \alpha \nabla l(w_k), \text{ or equivalently} \qquad (1)\\ \forall j\ w_{k+1,j} &\leftarrow w_{k,j} - \alpha \frac{\partial}{\partial w_{k,j}} l(w_k), \end{aligned}$$

where w_k are the weights from k^{th} iteration of the update, and $w_{k,j}$ is the j^{th} feature of w_k. Note that the above lines are equivalent because

$$\nabla l(w_k) = \left[\frac{\partial l(w_k)}{\partial w_{k,1}}, \frac{\partial l(w_k)}{\partial w_{k,2}}, \ldots, \frac{\partial l(w_k)}{\partial w_{k,m}}\right].$$

For this part of the assignment you will implement a gradient descent algorithm that performs the update in (1) N times using a step-size α.

As with the previous assignment, we have provided template code in the file `my_hw3.py` and a data set in the file `hw3_data.csv`. The template file contains code for loading the data set, initializing the weight vector w, running the gradient descent function, and then plotting the learning curve (the loss after each weight update). There is also functionality for running k-NN, but you can ignore this for now, and we will address it in the next section.

To implement gradient descent, complete the following function:

```python
def gradient_descent(X, Y, w0, alpha, num_iterations):
    losses = []
    w = np.copy(w0)

    # TODO complete this function

    return losses, w
```

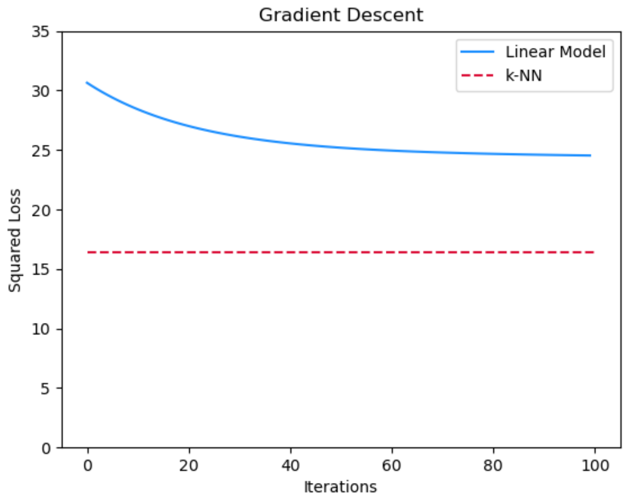

Figure 1: In this figure, the solid blue line indicates the loss function for each iteration of gradient descent, e.g., $l(w^k)$. The dashed red line represents the performance of the k-NN models where the number of neighbors is 100. This image is the output of a correct implementation of gradient descent using the hyperparameters, $\alpha = 10^{-5}$, $N = 100$.

This function should perform N iterations of (1) using the data set given by the feature vectors in X and labels in Y and return both a list of the losses for each iteration of gradient descent and the final weights found. More specifically, the objects returned are the list

$$[l(w_0), l(w_1), \ldots, l(w_N)],$$

and the weights w_N. Note that w_0 corresponds to the initial weight vector (`w0` in the source code). The input variables `w0`, `alpha`, and `num_iterations`, correspond to the vector w_0, step-size α, and the number of iterations N, respectively. We will be checking your implementation of this function for correct output using different data sets, weight vector initializations, step-sizes, and the number of iterations. Your implementation should match the output of our implementation up to floating-point precision. If your implementation is correct, you should see a plot similar to Figure 1, unless you have changed either the `alpha` or `num_iterations` variables in the function `main`.

There are many ways you can implement the gradient descent algorithm, and it is up to you to decide how best to do so. To help get you started, you might consider the following hints.

Hint: It might be helpful in your implementation to create a helper functions that compute the loss $l(w)$ and the gradient $\nabla l(w)$. We have provided function skeletons for these in the template code. The function `loss_function(X,Y,`w`)`, computes and returns the loss function on the data set. The function `loss_gradient(X,Y,`w`)`, computes and returns both the loss $l(w)$ and the gradient $\nabla l(w)$.

Hint: To test your implementation, you can create a new data set, where you can easily compute by hand the gradient and loss function. We suggest making a data set of only two data points and using feature vectors of length two, i.e., the data set contains the points (x_1, y_1) and (x_2, y_2) only, where $x_i = [x_{i,1}, x_{i,2}]$. You can make the labels whatever value you wish, but if you first pick a weight vector, $w^\star$, and compute by hand $f_{w^\star}(x_i)$ for each x_i, you can use the labels $y_i = f_{w^\star}(x_i)$. This data set will then have a nice property where $l(w^\star) = 0$, and it will make it easier to see if you are on the right track because w_k should get closer to $w^\star$ as k increases.

2 Comparing Linear Regression to k-NN (40 points)

There are many ways to compare either the predictive model's performance or the algorithms used to optimize a model. For this assignment, you will compare linear regression to k-NN based on the lowest possible loss each method can achieve. To find the minimum loss for each method, you will have to tune each algorithm's hyperparameters until you can no longer improve each model's performance. For linear regression, this means you have to tune the step-size α and number of iterations N for gradient descent. For k-NN, we provide a function to return the loss value for a data set, so you only have to tune the parameter k. These hyperparameters are listed in the `main()` function and represented by the variables `alpha` and `num_iterations` for gradient descent, and `k` for k-NN. There are **TODO** comments for each of these variables to make them easy to find. For both of these methods, you will receive full credit if the final loss is be below 14.0, and you are welcome to try to find the true minimum loss possible. An autograder will check the loss values by running your `main` function.

Hint: When tuning hyperparameters, we often begin with a guess about which values we think will be effective based on our understanding of the algorithm. After running the algorithm with these parameters, look at the resulting behavior of the algorithm and think about the impact that each hyperparameters has. This may allow you to deduce whether a parameter is too large or too small. Alternatively, you can try randomly perturbing the parameters until you get a feel for how the parameters impact the algorithm's behavior. For step sizes, we recommend beginning this process with $\alpha \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0\}$. For k we recommend beginning this process with k in the range 1 to 200.

Once you are satisfied with each algorithm's performance, you will create a report of your comparison using the LaTeX template file we provided. Submit this report as a `.pdf` file to gradescope. Your report should have four items in it: the near-optimal hyperparameter values you found, the corresponding loss values for each method, a learning curve plot, and the answers to the questions below. When recording the hyperparameters and the final loss value for each method, you must put these in Table 1. We have included this table in the `main.tex` template file for you to insert your answer. To find where you should place your hyperparameter and loss values, search for {value} and replace this with your value. Secondly, the learning curve plot should be the same as the one shown in Figure 1, but using the hyperparameters you found. When you run the `my_hw3.py` file, this plot will be automatically produced and saved in the file `learning_curve.png`. To add your plot to the report, replace the file `learning_curve.png` in the overleaf project's images folder with the same file output by the program after choosing your hyperparameter values. The image will then automatically update when you recompile `main.tex`.

Regression Models			
Model	Hyperparameters		Loss
Linear Regression	$\alpha = 10^{-5}$	$N = 100$	24.53
k-NN	$k = 100$		16.41

Table 1: Performances for each regression model.

Lastly, answer the questions below. When writing your answers, we have added the line

```
\textcolor{blue}{your answer here}
```

in the template file to indicate where you should put your answers. You may write your answer inside the { } by replacing the "your answer here" text. You may also notice that we split the questions into multiple pages. This page split is meant to make it easier for the graders by having answers on a single page. When submitting to gradescope, please mark which page of the PDF corresponds to each answer (this is done on gradescope when submitting).

Notice: Your responses will not be graded based on whether the answers are "correct". Instead, they will be graded based on whether you present reasonable arguments and show that you thought about the questions.

1. (4 points) How did you optimize the hyperparameters for each method? Summarize your process in 3–5 sentences.
2. (4 points) Which method did you find easier to tune and why? (3–5 sentences)
3. (4 points) Propose a different loss function (other than least squares), and explain why it might be a reasonable alternative. (No more than approximately 250 words.)
4. (4 points) Consider that there are n new data points, $(x_i, y_i), \dots$ and you are not allowed to refit w to this new data. You are then asked to make predictions of the labels using the model f_{w_N}. In general, are these predictions likely to be better or worse than the predictions on the data used to find w_N? When do you think these predictions will be better and when would they be worse? (No more than approximately 250 words.)
5. (4 points) Propose a real-world problem, that we have not discussed in class, for which you might apply a regression algorithm. Describe the problem, what the features and labels correspond to, how much data we might have, and what loss function might be appropriate. Can you think of any practical concerns (or ethical issues) when it comes to actually using regression algorithms for this proposed problem? (No more than approximately 250 words.)

3 Extra Credit

For a small amount of extra credit (3 points), implement gradient descent on the new loss function that you proposed in the written section, tune the hyperparameters, and provide a plot like Figure 1. We are not providing a code skeleton for the extra credit. You should submit a second python file containing your code for the extra credit (named `my_hw3_extra.py`).