Regression

During this week, we are exploring regression. Along the way, we will also deepen our understanding and experience with gradient descent, particularly as applied to regression. The previous notes on gradient descent will be useful, as well as the notes on regression.

In this lab, we will start by running code for solving regression problems and doing gradient descent on the mean square loss to develop an understanding for how it behaves. In the homework, you will implement all of the important functions necessary to create these demos.

Exploring gradient descent for regression and classification

In many problems, we want to predict a real value, such as the actual gas mileage of a car, or the concentration of some chemical. Luckily, we can use most of a mechanism we have already spent building up, and make predictors of the form:

$$y = \theta^T x + \theta_0$$

This is called a *linear regression* model.

We would like to learn a linear regression model from examples. Assume X is a d by n array (as before) but that Y is a 1 by n array of *floating-point* numbers (rather than +1 or -1). Given data (X, Y) we need to find θ, θ_0 that does a good job of making predictions on new data drawn from the same source.

We will approach this problem by formulating an objective function. There are many possible reasonable objective functions that implicitly make slightly different assumptions about the data, but they all typically have the form:

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0) \text{ (without regularization)}$$

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0) + \lambda R(\theta) \text{ (with regularization)}$$

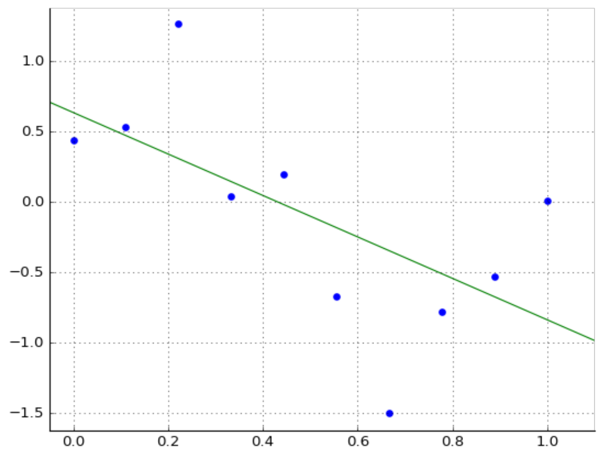
where L is our loss function and R is our regularization in terms of θ. For regression, we most frequently use *squared loss*, in which

$$L_s(x, y, \theta, \theta_0) = (\hat{y} - y)^2 = (\theta^T x + \theta_0 - y)^2$$

where our prediction is $\hat{y} = \theta^T x + \theta_0$ and y is our actual data.

We will come back to our regularization later...

In this lab, we will experiment with linear regression, gradient descent, and polynomial features. We will start with a simple data-set with one input feature and 10 training points, which is easy to visualize. In the plot below, the horizontal axis is our x value, and the vertical axis is the corresponding y value.



1) Least squares regression

In least squares regression problems, we assume that our objective function $J(\theta, \theta_0)$ comes without a regularization term; in other words,

$$J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0).$$

Recall from the lecture and notes that it is possible to solve a least-squares regression problem directly via the matrix algebra expression for the parameter vector θ in terms of the input data X and desired output vector Y. In this section, we will explore this *analytic* solution strategy.

For the rest of this lab, we will be computing solutions by transforming our one-dimensional input features with a polynomial basis. Specifically, we will be transforming our single input feature, which we will call x, into the vector $\phi(x) = (x^1, ..., x^k)$ if we are using a k-th order basis. This means that solutions to our regression problem will take the form

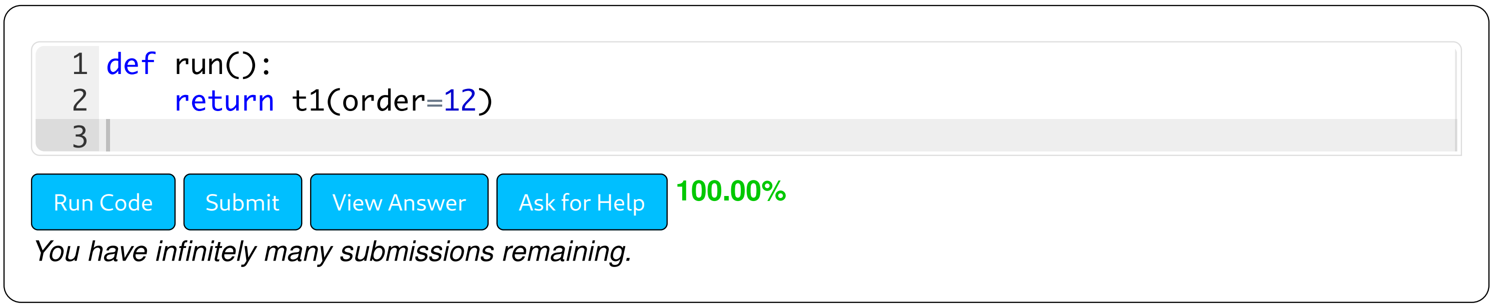
$$\theta^T\phi(x) + \theta_0 = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_k x^k$$

To answer the next set of problems, you are given the function `t1` to experiment with, which:

- Takes as input `order`, which is the order of the polynomial basis.
- Outputs θ and θ_0 that minimizes the regression objective $J(\theta, \theta_0)$.
- Finds θ and θ_0 **analytically** (we will explore the analytical solution in the homework).

In the codebox below, experiment with `t1` and try different values of order k to examine the effects of the order of the polynomial basis on the resulting solution. Based on your observations, answer the questions below the codebox.

To answer 1A and 1B below, you should try to answer the questions first without running `t1`, and then run `t1` to match with your expectations.


```
def run():
    return t1(order=12)

```

You have infinitely many submissions remaining.

1A) Consider the solution with the 0th-order basis (the regression polynomial described by only θ_0). What is the shape of the solution that you expect, and why? How does this solution depend on training data? Does this match what you observe in running `t1` with order 0?

1B) Polynomial order 9 is particularly interesting. What do you expect to be true about the solution, and why? (Hint: you are given 10 points in the training data). Does this match what you observe in running `t1` with order 9?

To answer the questions below, you should run the codebox above with various orders from 1 to 9 and observe plots and values of θ, θ_0. Make sure to click "submit".

1C) From your observation in the previous question, what is problematic if the polynomial order gets too big?

1D) Look at the values of θ and θ_0 you obtain, shown in the results. How does the magnitude of θ change with order? (You don't have to answer this quantitatively.)

1E) What polynomial order do you feel represents the hypothesis that will be the most predictive for new data?

1F) When we run 10-fold cross validation on this data set, varying the order from 0 to 8 (because we train on 9 training data points for each instance of 10-fold cross validation), we get the following mean squared error results:

```
[0.69206670716618535, 0.53820006043438084, 0.73424762793041687, 0.2835495578961193, 0.74338564580774558,
0.61422551802155112, 6.156711267187811, 356.8873742619765, 408.34678081302491]
```

What is the best order, based on this data? Does it agree with your previous answer?

1G) Abstractly, if we were to use a polynomial model with orders higher than 9 (e.g., 12), should it be possible for that polynomial to go through all the points?

1H) Now, actually run `t1` with orders higher than 9 (e.g., 12). Does the learned polynomial go through all the points? Remember that `t1` is fitting the polynomial analytically (directly using matrix inversion) as discussed in the notes on regression. Would we expect matrix inversion to work well for models with orders higher than 9 for this data?

2) Regularizing the parameter vector

Recall that we can add a regularization term $R(\theta)$ to the empirical risk. If we use a squared-norm regularizer, we get the so-called *ridge regression* objective:

$$J_{ridge}(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} L_s(x^{(i)}, y^{(i)}, \theta, \theta_0) + \lambda\|\theta\|^2$$

The procedure `t2(order, lam)` performs ridge regression on polynomial features of order `order` with regularization coefficient `lam`. In the resulting plots, `t2` draws the original least-squares solution in orange, and raws the regularized version in green.

Using a 9-th order polynomial feature function, experiment with λ. Look at the detailed results after submitting your code, paying attition to the visualization of the polynomial that was fit using your chosen λ.

```
def run():
    return t2(order=9, lam = 0.001)

```

Run Code Submit View Answer Ask for Help 100.00%

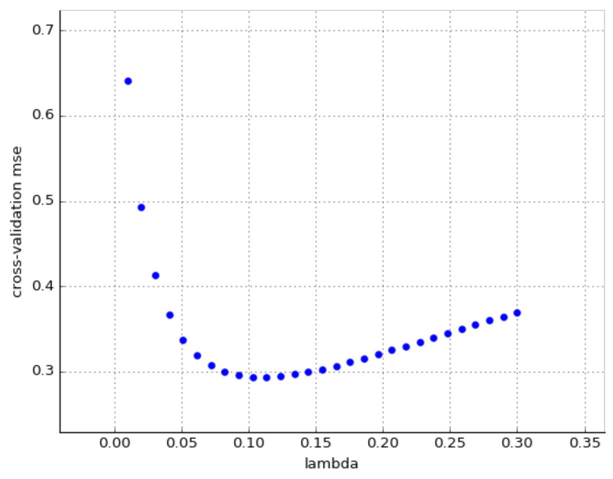
You have infinitely many submissions remaining.

2A) What happens to both $J_{ridge}(\theta, \theta_0)$ and the learned regression line, with very large (e.g., infinite) and very small (0) values of λ?

2B) If our goal is to solely minimize $\frac{1}{n}\sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0)$, what would be the best value of λ?

2C) What value of λ do you feel will give good performance on new data from the same source?

2D) When we run leave-one-out cross validation on this data set, using 9th order features, varying λ from 0.01 to 0.3, we get the following plot:



What is the best value of λ, in terms of generalization performance, based on this data? Does it agree with your previous answer? Is it the same as the best value for performance on the training set?

3) Gradient descent

Computing the analytic solution requires inverting a d by d matrix; as the size of the feature space increases, this becomes difficult. In addition, there are lots of other useful machine-learning models for which there is no closed-form analytic solution. So, we'll play with gradient descent here and see how it works. The procedure

```
t3(order, lam, step_size, max_iter)
```

performs gradient descent on the ridge regression objective using polynomial features of order `order`. You can specify the maximum number of iterations (we capped it at 10,000 to keep from killing the server) and the step size. It will print a convergence plot (objective value versus iteration number) and then show the solution it found in green and the analytic solution in orange for comparison.

Keeping $\lambda = 0$, experiment with this method for solving regression problems. (Note the difference in the plots when the solution diverges and when the solution converges very quickly.) Click 'Show/Hide Detailed Results' after submitting your code to view the convergence plot generated.

```
def run():
    return t3(order=9, lam=0, step_size= 0.325, max_iter = 10000)

```

Run Code | Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

3A) With `max_iter = 1000` in the code snippet above, what step sizes were needed to match the analytic solution almost exactly for 1st and 2nd order bases? (Use the convergence plot to help decide on whether the two curves "match almost exactly.")

3B) What step size allowed you to get similar curves to the analytical solution for 3rd order? (Use the convergence plot to help decide whether the two curves are "similar.")

3C) For 9th order, play around for a while (no more than 10 minutes) to see how close you can get to the analytic solution. How does it compare to the analytical solution?