(i) (**Logistic Regression and Batch Size**): Train the Logistic regression model with different batch size $b \in \{1, 50, 100, 200, 300\}$, learning rate $\eta = 10^{-5}$, and number of iterations $num_{iter} = 6000/b$. Train each model 50 or 100 times and average the test error for each value of batch size. Plot the test error as a function of the batch size. Which batch size gives the minimum test error?

Problem 5 (Programming Exercise: Polynomial Regression)

In this exercise, you will work through linear and polynomial regression. Our data consists of inputs $x_n \in \mathbb{R}$ and outputs $y_n \in \mathbb{R}, n \in \{1, \ldots, N\}$, which are related through a target function $y = f(x)$. Your goal is to learn a linear predictor $h_{\mathbf{w}}(x)$ that best approximates $f(x)$.

---

code and data

- code : `regression.py, Notebook.ipynb`
- data : `regression_train.csv`, `regression_test.csv`

Visualization

As we learned last week, it is often useful to understand the data through visualizations. For this data set, you can use a scatter plot to visualize the data since it has only two properties to plot (x and y).

(a) Visualize the training and test data using the `plot_data(...)` function. What do you observe? For example, can you make an educated guess on the effectiveness of linear regression in predicting the data?

Linear Regression

Recall that linear regression attempts to minimize the objective function

$$J(\mathbf{w}) = \sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_n) - y_n)^2.$$

In this problem, we will use the matrix-vector form where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \qquad \boldsymbol{X} = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{pmatrix}, \qquad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$$

and each instance $\mathbf{x}_n = \big(1, x_{n,1}, \ldots, x_{n,D}\big)^\top$.

In this instance, the number of input features $D = 1$.

Rather than working with this fully generalized, multivariate case, let us start by considering a simple linear regression model:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} = w_0 + w_1 x_1$$

`regression.py` contains the skeleton code for the class `Regression`. Objects of this class can be instantiated as `model = Regression (m)` where m is the degree of the polynomial feature vector where the feature vector for instance n, $\left(1, x_{n,1}, x_{n,1}^2, \ldots, x_{n,1}^m\right)^\top$. Setting $m = 1$ instantiates an object where the feature vector for instance n, $\left(1, x_{n,1}\right)^\top$.

(b) Note that to take into account the intercept term (w_0), we can add an additional "feature" to each instance and set it to one, e.g. $x_{i,0} = 1$. This is equivalent to adding an additional first column to $\boldsymbol{X}$ and setting it to all ones.

Modify `get_poly_features()` in `Regression.py` for the case $m = 1$ to create the matrix $\boldsymbol{X}$ for a simple linear model.

(c) Before tackling the harder problem of training the regression model, complete `predict()` in `Regression.py` to predict $\mathbf{y}$ from $\boldsymbol{X}$ and $\mathbf{w}$.

(d) One way to solve linear regression is through gradient descent (GD).

Recall that the parameters of our model are the w_j values. These are the values we will adjust to minimize $J(\mathbf{w})$. In gradient descent, each iteration performs the update

$$w_j \leftarrow w_j - 2\eta \sum_{n=1}^{N} \left(h_{\mathbf{w}}(\mathbf{x}_n) - y_n\right) x_{n,j} \quad \text{(simultaneously update } w_j \text{ for all } j\text{)}.$$

With each step of gradient descent, we expect our updated parameters w_j to come closer to the parameters that will achieve the lowest value of $J(\mathbf{w})$.

- As we perform gradient descent, it is helpful to monitor the convergence by computing the loss, *i.e.*, the value of the objective function J. Complete `loss_and_grad()` to calculate $J(\mathbf{w})$, and the gradient. Test your results by running the code in the main file `Notebook.ipnyb`. If you implement everything correctly, you should get the loss function around 4 and gradient approximately [-3.2, -10.5].

  We will use the following specifications for the gradient descent algorithm:

  - We run the algorithm for $10,000$ iterations.
  - We will use a fixed step size.
- So far, you have used a default learning rate (or step size) of $\eta = 0.01$. Try different $\eta = 10^{-4}, 10^{-3}, 10^{-1}$, and make a table of the coefficients and the final value of the objective function. How do the coefficients compare?

(e) In class, we learned that the closed-form solution to linear regression is

$$\mathbf{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \mathbf{y}.$$

Using this formula, you will get an exact solution in one calculation: there is no "loop until convergence" like in gradient descent.

- Implement the closed-form solution `closed_form()`.
- What is the closed-form solution? How do the coefficients and the cost compare to those obtained by GD? How quickly does the algorithm run compared to GD?

Polynomial Regression

Now let us consider the more complicated case of polynomial regression, where our hypothesis is

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) = w_0 + w_1 x + w_2 x^2 + \ldots + w^m x^m.$$

(f) Recall that polynomial regression can be considered as an extension of linear regression in which we replace our input matrix $\boldsymbol{X}$ with

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi(x_1)^\top \\ \phi(x_2)^\top \\ \vdots \\ \phi(x_N)^\top \end{pmatrix},$$

where $\phi(x)$ is a function such that $\phi_j(x) = x^j$ for $j = 0, \ldots, m$.

Update `gen_poly_features()` for the case when $m \geq 2$.

(g) For $m = \{0, \ldots, 10\}$, use the closed-form solver to determine the best-fit polynomial regression model on the training data, and with this model, calculate the loss on both the training data and the test data. Generate a plot depicting how loss varies with model complexity (polynomial degree) – you should generate a single plot with both training and test error, and include this plot in your writeup. Which degree polynomial would you say best fits the data? Was there evidence of under/overfitting the data? Use your plot to justify your answer.

Regularization

Finally, we will explore the role of regularization. For this problem, we will use ℓ_2-regularization so that our regularized objective function is

$$J(\theta) = \frac{1}{2} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n)^2 + \frac{\lambda}{2} ||\theta_{[1:m]}||^2,$$

again optimizing for the parameters θ.

(h) Modify `loss_and_grad()` to incorporate ℓ_2-regularization.

(i) Use your updated solver to find the coefficients that minimize the error for a tenth-degree polynomial ($m = 10$) given regularization factor $\lambda = 0, 10^{-8}, 10^{-7}, \ldots, 10^{-1}, 10^{0}$. Now use these coefficients to calculate the loss (unregularized) on both the training data and test data as a function of λ. Generate a plot depicting how the loss error varies with λ (for your x-axis, let $x = [1, 2, \ldots, 10]$ correspond to $\lambda = [0, 10^{-8}, 10^{-7}, \ldots, 10^{0}]$ so that λ is on a logistic scale, with regularization increasing as x increases). Which λ value appears to work best?