Assignment 2: Linear Regression

Machine Learning

Fall 2019

Prior Knowledge Utilized

- Basic machine learning terminology
- Supervised learning problem framing
- Partial derivatives and gradients, matrix-vector multiplication, and vector-vector multiplication

Learning Objectives

- Learn about the uses and limitations of linear regression.
- Learn how numerical optimization can be used to learn from data.
- Learn some useful mathematical tricks.
- Deepen understanding of basic machine learning terminology.

Recall: Mathematical background

In order to engage with this assignment, you'll want to make sure you are familiar with the following concepts (links to resources embedded below):

- vector-vector multiplication
  - Section 2.1 of Zico Kolter's Linear Algebra Review and Reference
- Matrix-vector multiplication
  - Section 2.2 of Zico Kolter's Linear Algebra Review and Reference
  - The first bits of the Khan academy video on Linear Transformations
- partial derivatives and gradients
  - Khan Academy videos on partial derivatives: intro, graphical understanding, and formal definition
  - Khan Academy video on Gradient

Recall: Supervised Learning Problem Setup

In the last class we saw the basic supervised learning problem setup.

Suppose you are given a training set of datapoints, $(\mathbf{x_1}, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$ where each $\mathbf{x_i}$ represents an element of an input space (e.g., a d-dimensional feature vector) and each y_i represents an element of an output space (e.g., a scalar target value). In the supervised learning setting, your goal is to determine a function $\hat{f}$ that maps from the input space to the output space.

We typically also assume that there is some loss function, ℓ, that determines the amount of loss that a particular prediction $\hat{y}_i$ incurs due to a mismatch with the actual output y_i. We can define the best possible model, $\hat{f}^\star$ as the one that minimizes these losses over the training set. This notion can be expressed with the following equation (note: that $\arg\min$ in the equation below just means the value that minimizes the expression inside of the $\arg\min$, e.g., $\arg\min_x (x-2)^2 = 2$, whereas $\min_x (x-2)^2 = 0$).

$$\hat{f}^\star = \arg\min_{\hat{f}} \sum_{i=1}^{n} \ell\left(\hat{f}(\mathbf{x_i}), y_i\right) \tag{1}$$

Recall: The Linear Regression Model

For linear regression we assume that our input points, $\mathbf{x_i}$, are d-dimensional vectors (each entry of these vectors can be though of as a feature), that our output points, y_i, are scalars, and that our prediction functions $\hat{f}$ are all of the form $\hat{f}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} = \sum_{i=1}^{d} w_i x_i$ for some vector of weights $\mathbf{w}$ (you could think of $\hat{f}$ as also taking $\mathbf{w}$ as an input, e.g., writing $\hat{f}(\mathbf{x}, \mathbf{w})$. When it's obvious what the value of $\mathbf{w}$ is, we'll leave it as an implicit input instead, e.g., writing $\hat{f}(\mathbf{x})$).

In the function, $\hat{f}$, the elements of the vector $\mathbf{w}$ represent weights that multiply various entries of the input. For instance, if an element of $\mathbf{w}$ is high, that means that as the corresponding element of $\mathbf{x}$ increases, the prediction that $\hat{f}$ generates for $\mathbf{x}$ would also increase (you may want to mentally think through other cases, e.g., what would happen is the element of $\mathbf{x}$ decreases, or what would happen if the entry of $\mathbf{w}$ was large and negative). The products of the weights and the features are then summed to arrive at an overall prediction.

Given this model, we can now define our very first machine learning algorithm: ordinary least squares (OLS)! In the ordinary least squares algorithm, we use our training set to select the $\mathbf{w}$ that minimizes the sum of squared differences between the model's predictions and the training outputs. Thinking back to the supervised learning problem setup, this corresponds to choosing $\ell(y, \hat{y}) = (y - \hat{y})^2$. Therefore, the OLS algorithm will use the training data to select the optimal value of $\mathbf{w}$ (called $\mathbf{w}^\star$), which minimizes the sum of squared differences between the model's predictions and the training outputs.

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \ell\left(\hat{f}(\mathbf{x_i}, \mathbf{w}), y_i\right) \tag{2}$$

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(\hat{f}(\mathbf{x_i}, \mathbf{w}) - y_i\right)^2 \tag{3}$$

$$= \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x_i} - y_i\right)^2 \tag{4}$$

While we haven't talked at all about how to find $\mathbf{w}^\star$, that will be the focus of a later part of this assignment, once we have $\mathbf{w}^\star$ we can predict a value for a new testing point, $\mathbf{x}$, by predicting that the corresponding (unknown) label, y, as $\hat{y} = \mathbf{w}^\top \mathbf{x}$. In this way, we have used the training data to learn how to make predictions about unseen data points, which is the hallmark of supervised machine learning!

1 *Linear Regression from the Bottom-up*

Now that you have some idea of how the algorithm behaves, let's figure out how to implement it. That is, how do we find the vector $\mathbf{w}$ that best fits a dataset?

1.1 *Linear regression with one variable*

Before handling the case where each $\mathbf{x}$ is a d-dimensional vector, we'll derive the algorithm for the simple case where $\mathbf{x}$ is a scalar (i.e., $d = 1$).

Exercise 1

(a) Given a dataset $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ (where each x_i and each y_i is a scalar) and a potential value of w, write an expression for the sum of squared errors between the model predictions, $\hat{f}$, and the targets, y_i.

(b) Compute the derivative the error function you found in part (a).

(c) Set the derivative to 0, and solve for $w^\star$. $w^\star$ corresponds to a critical point of your sum of squared errors function. We will show in a later assignment that this is critical point corresponds to a global minimum. In other words, this value of w is guaranteed to drive the sum of squared errors as low as possible.

We can start here and then motivate the mathematical tricks. asdf

1.2 Building our bag of mathematical tricks

Exercise 2 30 minutes

(a) Quadratic forms really nice Khan Academy video showing this

(b) FOIL method for matrix multiplication

(c) Using the definition of the gradient, show that $\nabla \mathbf{c}^\top \mathbf{x} = \mathbf{c}$ where the gradient is taken with respect to $\mathbf{x}$ and $\mathbf{c}$ is a vector of constants.

(d) Using the definition of the gradient, show that the $\nabla \mathbf{x}^\top \mathbf{A}\mathbf{x} = 2\mathbf{A}\mathbf{x}$ where the gradient is taken with respect to $\mathbf{x}$ and $\mathbf{A}$ is a *symmetric* *dxd* matrix of constants. If you'd like, you can utilize the fact (without deriving it yourself) that $\mathbf{x}^\top \mathbf{A}\mathbf{x} = \sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j a_{i,j}$.

2 Linear Regression with Multiple Variables

We might need to choose between this and a reading. If we do the reading, this will be done in class. Suppose you are given a training set of datapoints, $(\mathbf{x_1}, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$ where each $\mathbf{x_i}$ is a d-dimensional input vectors each y_i is a scalar representing a target output.

Exercise 3 (60 minutes)

Next, let's consider the more general case where $\mathbf{w}$ is a d-dimensional vector. In order to solve this problem, you'll be leveraging some of the new mathematical tricks you picked up earlier in the assignment. As you go through the problem, try as much as possible to treat vectors as first-class objects (e.g., work with the gradient instead of the individual partial derivatives).

(a) Given $\mathbf{w}$, write an expression for the sum of squared errors between each prediction $\hat{f}(\mathbf{x})$ and training output y_i. If your expression initially contains a summation, rewrite it as the inner product between some vector $\mathbf{v}$ and itself (i.e. $\mathbf{v}^\top \mathbf{v}$). In rewriting your expression as an inner-product, you may find it useful to refer to the matrix that contains all of your training data $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix}$. You may also want to expand your inner product using the FOIL method for matrices that we learned about earlier.

(b) Compute the gradient of the sum of squared errors that you found in part (a).

(c) Set the gradient to 0, and solve for $\mathbf{w}$. This value of $\mathbf{w}$ corresponds to a critical point of your sum of squared errors function. We will show in a later assignment that this is critical point corresponds to a global minimum. In other words, this value of $\mathbf{w}$ is guaranteed to drive the sum of squared errors as low as possible (note: you can assume that $\mathbf{X}^\top\mathbf{X}$ is invertible).

3 Implementation

TODO implement in Python through a guided notebook.

3.1 Sanity Checking

1. gradient check
2. optimality check
3. compare to known working solution

4 Experimenting with Data

Apply your code to smile detection (or some other task). This will start to get at the ML workflow.

5 Extensions

Regularization.

6 C+E

Todo this should show up somewhere.