CSM146, Winter 2018
Problem Set 2: Perceptron and Regression
Due Feb 6, 2018 at 11:59 pm

1 Perceptron [10 pts]

Design (specify $\boldsymbol{\theta}$ for) a two-input perceptron (with an additional bias or offset term) that computes the following boolean functions. Assume $T = 1$ and $F = -1$. If a valid perceptron exists, show that it is not unique by designing another valid perceptron (with a different hyperplane, not simply through normalization). If no perceptron exists, state why. **Solution:** For $\boldsymbol{x} = (1, x_1, x_2)$, a valid perceptron defined by $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2)$ requires that $y = \text{sign}(\boldsymbol{\theta}^T \boldsymbol{x})$.

[AND]

x_1	x_2	y
-1	-1	-1
-1	$+1$	-1
$+1$	-1	-1
$+1$	$+1$	$+1$

[XOR]

x_1	x_2	y
-1	-1	-1
-1	$+1$	$+1$
$+1$	-1	$+1$
$+1$	$+1$	-1

(a) **(5 pts)** AND **Solution:** One possible solution is $\boldsymbol{\theta} = (-1, 1, 1)$. Another possible solution is $\boldsymbol{\theta} = (-1, 0.5, 0.6)$. (there are other possible solutions)

(b) **(5 pts)** XOR **Solution:** No solution exists because the data is not linearly separable.

2 Logistic Regression [10 pts]

Consider the objective function that we minimize in logistic regression:

$$J(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \left[y_n \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) + (1 - y_n) \log \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right) \right]$$

Solution: Recall that we have $\sigma\prime(z) = \sigma(z)(1 - \sigma(z))$, and thus for $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$, we have:

$$\begin{aligned}
\frac{\partial h_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_k} &= \frac{\partial \sigma(\boldsymbol{\theta}^T \boldsymbol{x})}{\partial \theta_k} \\
&= \frac{\partial \sigma(\boldsymbol{\theta}^T \boldsymbol{x})}{\partial \boldsymbol{\theta}^T \boldsymbol{x}} \frac{\partial \left(\boldsymbol{\theta}^T \boldsymbol{x}\right)}{\partial \theta_k} \\
&= \sigma(\boldsymbol{\theta}^T \boldsymbol{x})(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x}) \frac{\partial \left(\sum_j \theta_j x_j\right)}{\partial \theta_k} \\
&= \sigma(\boldsymbol{\theta}^T \boldsymbol{x})(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x}) x_k \\
&= h_{\boldsymbol{\theta}}(\boldsymbol{x})(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) x_k
\end{aligned} \tag{1}$$

Parts of this assignment are adapted from course material by Andrew Ng (Stanford), Jenna Wiens (UMich) and Jessica Wu (Harvey Mudd).

This latter fact is very useful to make the following derivations.

(a) **(10 pts)** Find the partial derivatives $\frac{\partial J}{\partial \theta_j}$. **Solution:**

$$\begin{aligned}
\frac{\partial J}{\partial \theta_j} &= \frac{\partial \left(-\sum_{n=1}^{N} \left[y_n \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) + (1-y_n) \log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right)\right]\right)}{\partial \theta_j} \\
&= -\sum_{n=1}^{N} \frac{\partial [y_n \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) + (1-y_n) \log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right)]}{\partial \theta_j} \\
&= -\sum_{n=1}^{N} \left(\frac{\partial \left[y_n \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right]}{\partial \theta_j} + \frac{\partial \left[(1-y_n) \log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right)\right]}{\partial \theta_j} \right) \\
&= -\sum_{n=1}^{N} \left(y_n \frac{\partial \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)}{\partial \theta_j} + (1-y_n) \frac{\partial \log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right)}{\partial \theta_j} \right)
\end{aligned}$$

To compute the partial derivatives, we use the chain rule and identity 1

$$\begin{aligned}
\frac{\partial \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)}{\partial \theta_j} &= \frac{\partial \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)}{\partial h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)} \frac{\partial h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)}{\partial \theta_j} \\
&= \frac{1}{h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)} h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)) x_j \\
&= (1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)) x_j \qquad (2)
\end{aligned}$$

Similarly

$$\frac{\partial \log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right)}{\partial \theta_j} = -h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) x_j \qquad (3)$$

Using Equations 2 and 3

$$\begin{aligned}
\frac{\partial J}{\partial \theta_j} &= -\sum_{n=1}^{N} \left(y_n (1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)) x_j - (1-y_n)\, h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) x_j \right) \\
&= -\sum_{n=1}^{N} \left(y_n - h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) \right) x_{n,j} \\
&= \sum_{n=1}^{N} \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) - y_n \right) x_{n,j}
\end{aligned}$$

(it is fine if the formulation is not simplified.)

3 Locally Weighted Linear Regression [10 pts]

Consider a linear regression problem in which we want to "weight" different training instances differently because some of the instances are more important than others. Specifically, suppose we want to minimize

$$J(\theta_0, \theta_1) = \sum_{n=1}^{N} w_n \left(\theta_0 + \theta_1 x_{n,1} - y_n \right)^2.$$

Here $w_n > 0$. In class, we worked out what happens for the case where all the weights (the w_n's) are the same. In this problem, we will generalize some of those ideas to the weighted setting.

(a) **(6 pts)** Calculate the gradient by computing the partial derivatives of J with respect to each of the parameters (θ_0, θ_1). **Solution:**

$$\begin{aligned}\frac{\partial J}{\partial \theta_0} &= \sum_{n=1}^{N} 2w_n \left(\theta_0 + \theta_1 x_{n,1} - y_n\right) \\ \frac{\partial J}{\partial \theta_1} &= \sum_{n=1}^{N} 2w_n x_{n,1} \left(\theta_0 + \theta_1 x_{n,1} - y_n\right)\end{aligned}$$

(b) **(4 pts)** Set each partial derivatives to 0 and solve for θ_0 and θ_1 to obtain values of (θ_0, θ_1) that minimize J. **Solution:**

$$\begin{aligned}\frac{\partial J}{\partial \theta_0} &= 0 \\ \Rightarrow \sum_{n=1}^{N} 2w_n \left(\theta_0 + \theta_1 x_{n,1} - y_n\right) &= 0 \\ \Rightarrow \sum_{n=1}^{N} w_n \left(\theta_0 + \theta_1 x_{n,1} - y_n\right) &= 0 \\ \Rightarrow \sum_{n=1}^{N} w_n \theta_0 + \sum_{n=1}^{N} w_n \theta_1 x_{n,1} - \sum_{n=1}^{N} w_n y_n &= 0 \\ \Rightarrow \left(\sum_{n=1}^{N} w_n\right) \theta_0 + \left(\sum_{n=1}^{N} w_n x_{n,1}\right) \theta_1 &= \sum_{n=1}^{N} w_n y_n \end{aligned} \tag{4}$$

$$\begin{aligned}\frac{\partial J}{\partial \theta_1} &= 0 \\ \Rightarrow \sum_{n=1}^{N} 2w_n x_{n,1} \left(\theta_0 + \theta_1 x_{n,1} - y_n\right) &= 0 \\ \Rightarrow \sum_{n=1}^{N} w_n x_{n,1} \left(\theta_0 + \theta_1 x_{n,1} - y_n\right) &= 0 \\ \Rightarrow \sum_{n=1}^{N} w_n x_{n,1} \theta_0 + \sum_{n=1}^{N} w_n \theta_1 {x_{n,1}}^2 - \sum_{n=1}^{N} w_n x_{n,1} y_n &= 0 \\ \Rightarrow \left(\sum_{n=1}^{N} w_n x_{n,1}\right) \theta_0 + \left(\sum_{n=1}^{N} w_n {x_{n,1}}^2\right) \theta_1 &= \sum_{n=1}^{N} w_n y_n x_{n,1} \end{aligned} \tag{5}$$

Since $w_n > 0$, we divide throughout by $\left(\sum_{n=1}^{N} w_n\right)$. We also define the weighted means of

averages:

$$\begin{aligned}
\overline{x} &= \frac{\sum_{n=1}^{N} w_n x_{n,1}}{\sum_{n=1}^{N} w_n} \\
\overline{y} & \frac{\sum_{n=1}^{N} w_n y_n}{\sum_{n=1}^{N} w_n} \\
\overline{x^2} &= \frac{\sum_{n=1}^{N} w_n {x_{n,1}}^2}{\sum_{n=1}^{N} w_n} \\
\overline{xy} &= \frac{\sum_{n=1}^{N} w_n y_n x_{n,1}}{\sum_{n=1}^{N} w_n}
\end{aligned}$$

We then rewrite Equation 4 and 5

$$\theta_0 + \overline{x}\theta_1 = \overline{y} \tag{6}$$

$$\overline{x}\theta_0 + \overline{x^2}\theta_1 = \overline{xy} \tag{7}$$

Substituting for θ_0 from Equation 6 into Equation 7

$$\overline{x}\left(\overline{y} - \overline{x}\theta_1\right) + \overline{x^2}\theta_1 = \overline{xy} \tag{8}$$

Re-arranging to solve for θ_1:

$$\widehat{\theta_1} = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - (\overline{x})^2} \tag{9}$$

Plugging Equation 9 back into Equation 6, we can solve for θ_0

$$\widehat{\theta_0} = \overline{y} - \overline{x}\widehat{\theta_1} \tag{10}$$

The solution for weighted linear regression is similar to the ordinary least squares except that we replace unweighted averages with weighted averages.

(it is fine if the formulation is not simplified.)

4 Understanding Linear Separability [25 pts]

Definition 1 (Linear Program) *A* linear program *can be stated as follows:*

Let A be an $m \times n$ real-valued matrix, $\vec{b} \in \mathbb{R}^m$, and $\vec{c} \in \mathbb{R}^n$. Find a $\vec{t} \in \mathbb{R}^n$, that minimizes the linear function

$$\begin{aligned} & z(\vec{t}) = \vec{c}^{\,T}\vec{t} \\ \textit{subject to} \quad & A\vec{t} \geq \vec{b} \end{aligned}$$

In the linear programming terminology, $\vec{c}$ is often referred to as the *cost vector* and $z(\vec{t})$ is referred to as the *objective function.*[1] We can use this framework to define the problem of learning a linear discriminant function.[2]

The Learning Problem:[3] Let $\vec{x_1}, \vec{x_2}, \ldots, \vec{x_m}$ represent m samples, where each sample $\vec{x_i} \in \mathbb{R}^n$ is an n-dimensional vector, and $\vec{y} \in \{-1, 1\}^m$ is an $m \times 1$ vector representing the respective labels of each of the m samples. Let $\vec{w} \in \mathbb{R}^n$ be an $n \times 1$ vector representing the weights of the linear discriminant function, and θ be the threshold value.

We *predict* $\vec{x_i}$ to be a *positive* example if $\vec{w}^T\vec{x_i} + \theta \geq 0$. On the other hand, we *predict* $\vec{x_i}$ to be a *negative* example if $\vec{w}^T\vec{x_i} + \theta < 0$.

We hope that the learned linear function can separate the data set. That is,

$$y_i = \begin{cases} 1 & \text{if } \vec{w}^T\vec{x_i} + \theta \geq 0 \\ -1 & \text{if } \vec{w}^T\vec{x_i} + \theta < 0. \end{cases} \tag{11}$$

In order to find a good linear separator, we propose the following linear program:

$$\begin{aligned} \min \quad & \delta \\ \text{subject to} \quad & y_i(\vec{w}^T\vec{x_i} + \theta) \geq 1 - \delta, \qquad \forall(\vec{x_i}, y_i) \in D \\ & \delta \geq 0 \end{aligned} \tag{12}$$

where D is the data set of all training examples.

[1] Note that you don't need to really know how to solve a linear program since we will use Matlab to obtain the solution (although you may wish to brush up on Linear Algebra). Also see the appendix for more information on linear programming.

[2] This discussion closely parallels the linear programming representation found in *Pattern Classification*, by Duda, Hart, and Stork.

[3] Note that the notation used in the Learning Problem is **unrelated** to the one used in the Linear Program definition. You may want to figure out the correspondence.

(a) **(5 pts)** A data set $D = \{(\vec{x_i}, y_i)\}_{i=1}^m$ that satisfies condition (1) above is called *linearly separable*. Show that if D is linearly separable, there is an optimal solution to the linear program (2) with $\delta = 0$ **Solution:**

(i) Linear Separability $\Rightarrow \delta = 0$:

The separability property implies that there exists a hyperplane $\vec{v}^T\vec{x} + \rho$ such that

$$\min_{\substack{(\vec{x},y)\in D \\ y=1}} (\vec{v}^T\vec{x} + \rho) \quad \geq \quad 0 \quad > \quad \max_{\substack{(\vec{x},y)\in D \\ y=-1}} (\vec{v}^T\vec{x} + \rho)$$

Let $\vec{x_i}$ to be the positive sample that is closest to the hyperplane $\vec{v}^T\vec{x} + \rho$, and

$$p^+ = \min_{\substack{(\vec{x},y)\in D \\ y=1}} (\vec{v}^T\vec{x} + \rho) = (\vec{v}^T\vec{x_i} + \rho)$$

Similarly, let $\vec{x_j}$ to be the negative sample that is closest to the hyperplane $\vec{v}^T\vec{x} + \rho$, and

$$p^- = \max_{\substack{(\vec{x},y)\in D \\ y=-1}} (\vec{v}^T\vec{x} + \rho) = (\vec{v}^T\vec{x_j} + \rho)$$

Note that from the definition of separability, we know that p^+ and p^- exist and $p^+ \geq 0 > p^-$.

Step 1: Shifting the hyperplane. We need to make sure the target hyperplane does not exactly lie on p^+ and p^- in order to guarantee a minimum distance. Since $p^+ \geq 0 > p^-$, there exist $\eta \geq 0$ such that $p^+ - \eta \geq 0 > p^- - \eta$. Hence, for some values of η, the hyperplane $\vec{v}^T\vec{x} + \rho - \eta = 0$ also separates D. We will find the value of η for which the hyperplane $\vec{v}^T\vec{x} + \rho - \eta = 0$ is equi-distant from $\vec{x_i}$ and $\vec{x_j}$ and separates D.

The value of η can be obtained as follows:

$$\begin{aligned} \frac{|\vec{v}^T\vec{x_i} + \rho - \eta|}{\|\vec{v}\|} &= \frac{|\vec{v}^T\vec{x_j} + \rho - \eta|}{\|\vec{v}\|}, \\ \vec{v}^T\vec{x_i} + \rho - \eta &= -(\vec{v}^T\vec{x_j} + \rho - \eta), \\ p^+ - \eta &= -p^- + \eta. \end{aligned} \tag{1}$$

This implies that $\eta = \frac{p^+ + p^-}{2}$. We can easily verify that for such η, $p^+ - \eta \geq 0 > p^- - \eta$, and hence the resulting hyperplane $\vec{v}^T\vec{x} + \rho - \eta = 0$ separates D.

Step 2: Normalizing the hyperplane. With the new hyperplane $\vec{v}^T\vec{x} + \rho - \eta = 0$,

$$\min_{\substack{(\vec{x},y)\in D \\ y=1}} (\vec{v}^T\vec{x} + \rho - \eta) = \frac{p^+ - p^-}{2}$$

and

$$\max_{\substack{(\vec{x},y)\in D \\ y=-1}} (\vec{v}^T\vec{x} + \rho - \eta) = \frac{p^- - p^+}{2}$$

$$\therefore y(\vec{v}^T\vec{x} + \rho - \eta) \geq \frac{p^+ - p^-}{2} \qquad \forall (\vec{x}, y) \in D$$

Given that $p^+ > p^-$ and $\eta = \frac{p^+ + p^-}{2}$, let $\eta' = \frac{p^+ - p^-}{2}$ and by setting $\vec{w} = \frac{\vec{v}}{\eta'}$, $\theta = \frac{\rho - \eta}{\eta'}$, and $\delta = 0$,

$$y(\vec{w}^T\vec{x} + \theta) \geq 1 - \delta, \quad \forall (\vec{x}, y) \in D$$

Note that there may be different answers based on the different shifting options. Give full score if the students do the scaling.

(b) **(5 pts)** Now show that there is an optimal solution with $\delta = 0$, then D is linearly separable. **Solution:**

If there exists a hyperplane $\vec{w}^T\vec{x} + \theta$ such that

$$y(\vec{w}^T\vec{x} + \theta) \geq 1, \quad \forall(\vec{x}, y) \in D$$

It is trivial to show that

$$\begin{aligned} & (\vec{w}^T\vec{x} + \theta) \geq 1 \quad \geq \quad 0, \qquad \forall(\vec{x}, y) \in D, y = 1 \\ \text{and} \quad & (\vec{w}^T\vec{x} + \theta) \leq -1 \quad < \quad 0, \qquad \forall(\vec{x}, y) \in D, y = -1 \end{aligned}$$

(c) **(5 pts)** What can we say about the linear separability of the data set if there exists a hyperplane that satisfies condition (2) with $\delta > 0$? **Solution:**

When $\delta > 0$: Now consider the value of δ. Note that if $1 - \delta > 0$, we can apply similar argument and show that the data set is linear separable. If $\delta \geq 1$, we are not sure if the data set is separable or not. If the *minimal* $\delta \geq 1$, then the data set is not separable.

(d) **(5 pts)** An alternative LP formulation to (2) may be

$$\begin{aligned} \min \quad & \delta \\ \text{subject to} \quad & y_i(\vec{w}^T\vec{x_i} + \theta) \geq -\delta, \qquad \forall(\vec{x_i}, y_i) \in D \\ & \delta \geq 0 \end{aligned}$$

Find the optimal solution to this formulation (independent of D) to illustrate the issue with such a formulation. **Solution:**

The trivial solution is

$$\vec{w} = \vec{0}, \quad \theta = 0, \quad \delta = 0.$$

This solution is optimal because 1) the constraints are satisfied, and 2) zero is the best value we can get for δ. We do not use this formulation because the solver might give us this optimal solution which does not give us a hyperplane at all. Note that there exists other optimal solutions for this formulation.

(e) **(5 pts)** Let $\vec{x_1} \in \mathbb{R}^n$, $\vec{x_1}^T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ and $y_1 = 1$. Let $\vec{x_2} \in \mathbb{R}^n$, $\vec{x_2}^T = \begin{bmatrix} -1 & -1 & -1 \end{bmatrix}$ and $y_2 = -1$. The data set D is defined as

$$D = \{(\vec{x_1}, y_1), (\vec{x_2}, y_2)\}.$$

Consider the formulation in (2) applied to D. What are possible optimal solutions?

Solution:

Given that there are only two samples in the data set, the data set is separable. Therefore, the optimal δ is 0. Now our job becomes finding $\vec{w}$ and θ that satisfy the following constraints:

$$\begin{aligned} w_1 + w_2 + w_3 + \theta &\geq 1 \\ -(-w_1 - w_2 + -w_3 + \theta) &\geq 1 \\ \\ \therefore \quad w_1 + w_2 + w_3 &\geq 1 + |\theta| \end{aligned}$$

Hence, $(\vec{w}, \theta, \delta)$ is the optimal solution if $\delta = 0$ and $w_1 + w_2 + w_3 \geq 1 + |\theta|$.

Solution: Refer q4 in https://courses.engr.illinois.edu/cs446/sp2015/Homework/HW2/hw2-solution.pdf

5 Implementation: Polynomial Regression [45 pts]

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 $h_{\boldsymbol{\theta}}(x)$ that best approximates $f(x)$. But this time, rather than using `scikit-learn`, we will further open the "black-box", and you will implement the regression model!

---

code and data

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

This is likely the first time that many of you are working with `numpy` and matrix operations within a programming environment. For the uninitiated, you may find it useful to work through a `numpy` tutorial first.[4] Here are some things to keep in mind as you complete this problem:

- If you are seeing many errors at runtime, inspect your matrix operations to make sure that you are adding and multiplying matrices of compatible dimensions. Printing the dimensions of variables with the `X.shape` command will help you debug.
- When working with `numpy` arrays, remember that `numpy` interprets the `*` operator as element-wise multiplication. This is a common source of size incompatibility errors. If you want matrix multiplication, you need to use the `dot` function in Python. For example, `A*B` does element-wise multiplication while `dot(A,B)` does a matrix multiply.
- Be careful when handling `numpy` vectors (rank-1 arrays): the vector shapes $1 \times N$, $N \times 1$, and N are all different things. For these dimensions, we follow the the conventions of `scikit-learn`'s `LinearRegression` class[5]. Most importantly, unless otherwise indicated (in the code documentation), both column and row vectors are rank-1 arrays of shape N, not rank-2 arrays of shape $N \times 1$ or shape $1 \times N$.

Visualization [5 pts]

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) **(5 pts)** 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?

Solution: Neither data set shows a linear relationship. We would expect simple linear regression to be a poor predictor, but polynomial regression might work.

[4] Try out `SciPy`'s tutorial (http://wiki.scipy.org/Tentative_NumPy_Tutorial), or use your favorite search engine to find an alternative. Those familiar with Matlab may find the "Numpy for Matlab Users" documentation (http://wiki.scipy.org/NumPy_for_Matlab_Users) more helpful.

[5] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Linear Regression [25 pts]

Recall that linear regression attempts to minimize the objective function

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) - y_n\right)^2.$$

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

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \qquad \boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{pmatrix}, \qquad \boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_D \end{pmatrix}$$

and each instance $\boldsymbol{x}_n = \left(1, x_{n,1}, \ldots, x_{n,D}\right)^T$.

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:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} = \theta_0 + \theta_1 x_1$$

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

(b) **(2 pts)** Note that to take into account the intercept term (θ_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 `PolynomialRegression.generate_polynomial_features(...)` to create the matrix $\boldsymbol{X}$ for a simple linear model. **Solution:** No answer is needed.

(c) **(2 pts)** Before tackling the harder problem of training the regression model, complete `PolynomialRegression.predict(...)` to predict $\boldsymbol{y}$ from $\boldsymbol{X}$ and $\boldsymbol{\theta}$. **Solution:** No answer is needed.

(d) **(10 pts)** One way to solve linear regression is through gradient descent (GD).

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

$$\theta_j \leftarrow \theta_j - 2\alpha \sum_{n=1}^{N} \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) - y_n\right) x_{n,j} \quad \text{(simultaneously update } \theta_j \text{ for all } j\text{)}.$$

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

- **(3 pts)** As we perform gradient descent, it is helpful to monitor the convergence by computing the cost, *i.e.*, the value of the objective function J. Complete `PolynomialRegression.cost(...)` to calculate $J(\boldsymbol{\theta})$.

  If you have implemented everything correctly, then the following code snippet should return 40.234.

```
train_data = load_data('regression_train.csv')
model = PolynomialRegression()
model.coef_ = np.zeros(2)
model.cost(train_data.X, train_data.y)
```

  Solution: No answer is needed.

- **(2 pts)** Next, implement the gradient descent step in `PolynomialRegression.fit_GD(...)`. The loop structure has been written for you, and you only need to supply the updates to $\boldsymbol{\theta}$ and the new predictions $\hat{y} = h_{\boldsymbol{\theta}}(\boldsymbol{x})$ within each iteration.

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

  - We run the algorithm for $10,000$ iterations.
  - We terminate the algorithm ealier if the value of the objective function is unchanged across consecutive iterations.
  - We will use a fixed step size.

  Solution: No answer is needed.

- **(5 pts)** So far, you have used a default learning rate (or step size) of $\eta = 0.01$. Try different $\eta = 10^{-4}, 10^{-3}, 10^{-2}, 0.0407$, and make a table of the coefficients, number of iterations until convergence (this number will be $10,000$ if the algorithm did not converge in a smaller number of iterations) and the final value of the objective function. How do the coefficients compare? How quickly does each algorithm converge? **Solution:** See Table 1. If you choose a learning rate that is too large, the algorithm diverges. If you choose a learning rate that is too small, the coefficients are closer to the closed-form solution, but the algorithm takes a very long time to converge.

η	θ	iterations	cost
10^{-4}	$[2.2704; -2.4606]$	$10,000$	4.0864
10^{-3}	$[2.4464; -2.8163]$	$7,077$	3.91258
10^{-2}	$[2.4464; -2.8163]$	766	3.91258
0.0407	$[-9.4 \times 10^{18}; -4.65 \times 10^{18}]$	$10,000$	2.71×10^{39}

Table 1: Optimal coefficients, number of iterations and cost function for varying step sizes.

(e) **(6 pts)** In class, we learned that the closed-form solution to linear regression is

$$\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

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

- **(1 pts)** Implement the closed-form solution `PolynomialRegression.fit(...)`. **Solution:** No answer is needed.

- **(5 pts)** 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? **Solution:** The coefficients are $[2.4464; -2.8164]$ and the cost is 3.912576. For GD with smaller step sizes, the coefficients are closer to the closed-form solution. In this particular instance, the closed-form solution is faster than gradient descent.

  [3 pt for comparison of coefficients (full credit as long as solution states that the two approaches yield same solution for appropriate choices of model parameters, e.g. learning rates), 2 pt for comparison of runtime]

(f) **(5 pts)** Finally, set a learning rate η for GD that is a function of k (the number of iterations) (use $\eta_k = \frac{1}{1+k}$) and converges to the same solution yielded by the closed-form optimization (minus possible rounding errors). Update `PolynomialRegression.fit_GD(...)` with your proposed learning rate. How long does it take the algorithm to converge with your proposed learning rate? **Solution:** $\eta_k = \frac{1}{1+k}$, where k is the number of iterations of the outer loop converges to the same solution yielded by the closed form optimization. It takes 1511 iterations for the algorithm to converge to $[2.4464; -2.8163]$, with this learning rate. There are multiple other possible solutions.

[5 pt for any function that decreases as k increases]

Polynomial Regression[15 pts]

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

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \phi(\boldsymbol{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta^m x^m.$$

(g) **(2 pts)** Recall that polynomial regression can be considered as an extension of linear regression in which we replace our input matrix $\boldsymbol{X}$ with

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

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

Update `PolynomialRegression.generate_polynomial_features(...)` to create an $m+1$ dimensional feature vector for each instance. **Solution:** No answer is needed.

(h) **(3 pts)** Given N training instances, it is always possible to obtain a "perfect fit" (a fit in which all the data points are exactly predicted) by setting the degree of the regression to $N-1$. Of course, we would expect such a fit to generalize poorly. In the remainder of this problem, you will investigate the problem of overfitting as a function of the degree of the polynomial, m. To measure overfitting, we will use the Root-Mean-Square (RMS) error, defined as

$$E_{RMS} = \sqrt{J(\boldsymbol{\theta})/N},$$

where N is the number of instances.[6]

[6]Note that the RMSE as defined is a biased estimator. To obtain an unbiased estimator, we would have to divide by $n-k$, where k is the number of parameters fitted (including the constant), so here, $k = m+1$.

Why do you think we might prefer RMSE as a metric over $J(\boldsymbol{\theta})$? **Solution:** RMSE has multiple advantages: (1) [3 pts] it is more comparable across different data set sizes since it normalizes by the number of instances; (2) [2 pts] it represents the sample standard deviation of the residuals and is therefore on the same scale as the predictions (compared to the the MSE, which represents the sample variance).

Implement `PolynomialRegression.rms_error(...)`. **Solution:** No answer is needed.

(i) **(10 pts)** 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 RMSE on both the training data and the test data. Generate a plot depicting how RMSE 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. **Solution:** See Figure 1. 4^{th}, 5^{th}, and 6^{th} degree polynomials fit the data best. For $m > 8$, there is clear evidence of overfitting – the test error increases greatly but training error decreases. For $m < 3$, there is evidence of underfitting – the training error is very high.

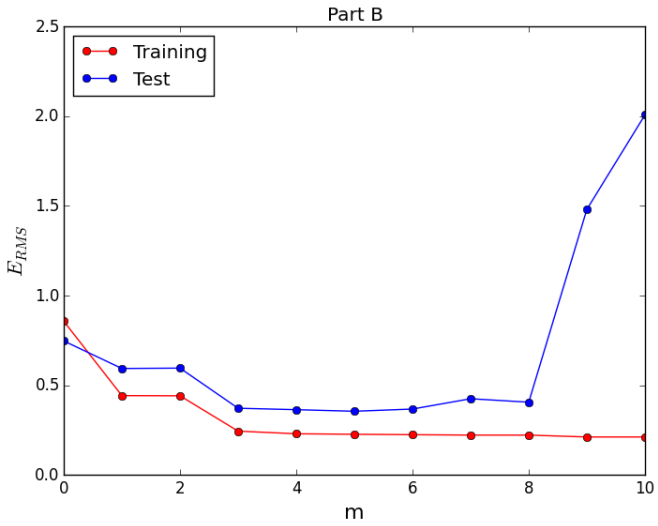

Figure 1: Training and testing error as a function of model complexity.