CM146, Winter 2024
Problem Set 2: Perceptron and Regression
Due February 16, 2024 at 11:59 pm

Sophia Sharif

Submission

- Submit your solutions electronically on the course Gradescope site as PDF files.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.

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

1 Perceptron [2 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.

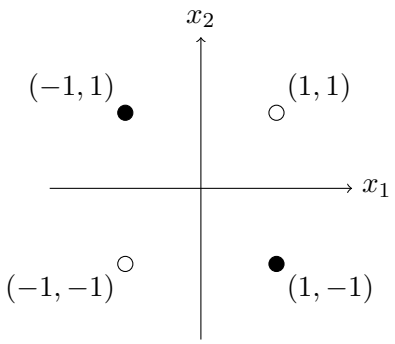
(a) **(1 pts)** OR

Solution: The data for the OR function is linearly separable, so there exist many valid perceptrons. For example, $\boxed{\boldsymbol{\theta_1} = \langle -1, 2, 2 \rangle}$ and $\boxed{\boldsymbol{\theta_2} = \langle -1, 3, 3 \rangle}$ with input vector $\boldsymbol{x} = \langle 1, x_1, x_2 \rangle$ are both valid solutions.

x_1	x_2	$\boldsymbol{\theta_1}^T \boldsymbol{x}$	$\boldsymbol{\theta_2}^T \boldsymbol{x}$	OR(x_1, x_2)
0	0	-1	-1	-1
0	1	1	2	1
1	0	1	2	1
1	1	3	5	1

(b) **(1 pts)** XOR

Solution: The data for the XOR function is not linearly separable, so no valid perceptron exists. As evident in the graph below, there is no line that can separate the two classes.



2 Logistic Regression [10 pts]

Consider the objective function that we minimize in logistic regression:

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

where $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$.

(a) **(2 pts)** Find the partial derivatives $\frac{\partial J}{\partial \theta_j}$.
Solution: For a single data point:

$$\begin{aligned}
\frac{\partial J}{\partial \theta_j} &= -\left[y_n \frac{1}{h_{\boldsymbol{\theta}}(\boldsymbol{x_n})} \frac{\partial h_{\boldsymbol{\theta}}(\boldsymbol{x_n})}{\partial \theta_j} - (1-y_n) \frac{1}{1-h_{\boldsymbol{\theta}}(\boldsymbol{x_n})} \frac{\partial h_{\boldsymbol{\theta}}(\boldsymbol{x_n})}{\partial \theta_j} \right] \\
&= -\frac{\partial h_{\boldsymbol{\theta}}(\boldsymbol{x_n})}{\partial \theta_j} \left[\frac{y_n - h_{\boldsymbol{\theta}}(\boldsymbol{x_n})}{h_{\boldsymbol{\theta}}(\boldsymbol{x_n})\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x_n})\right)} \right]. \\
&= -\frac{h_{\boldsymbol{\theta}}(\boldsymbol{x_n})\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x_n})\right)}{h_{\boldsymbol{\theta}}(\boldsymbol{x_n})\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x_n})\right)} \left[y_n - h_{\boldsymbol{\theta}}(\boldsymbol{x_n}) \right] \frac{\partial \left(\boldsymbol{\theta}^T \boldsymbol{x}\right)}{\partial \theta_j} \\
&= -\left[y_n - h_{\boldsymbol{\theta}}(\boldsymbol{x_n}) \right] x_{n,j}.
\end{aligned}$$

So, the partial derivative of the objective function with respect to θ_j is

$$\boxed{\frac{\partial J}{\partial \theta_j} = -\sum_{n=1}^{N} \left[y_n - h_{\boldsymbol{\theta}}(\boldsymbol{x_n}) \right] x_{n,j}}$$

(b) **(3 pts)** Find the partial second derivatives $\frac{\partial^2 J}{\partial \theta_j \partial \theta_k}$ and show that the Hessian (the matrix $\boldsymbol{H}$ of second derivatives with elements $H_{jk} = \frac{\partial^2 J}{\partial \theta_j \partial \theta_k}$) can be written as $\boldsymbol{H} = \sum_{n=1}^{N} h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right)\boldsymbol{x}_n \boldsymbol{x}_n^T$.
Solution: For a single data point:

$$\begin{aligned}
\frac{\partial^2 J}{\partial \theta_j \partial \theta_k} &= -\frac{\partial}{\partial \theta_j} \left[y_n - h_{\boldsymbol{\theta}}(\boldsymbol{x_n}) \right] x_{n,k} \\
&= -\left(0 - h_{\boldsymbol{\theta}}(\boldsymbol{x_n})\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x_n})\right) x_{n,j} x_{n,k} \right)
\end{aligned}$$

So, the partial second derivative of the objective function with respect to θ_j and θ_k is

$$\frac{\partial^2 J}{\partial \theta_j \partial \theta_k} = \sum_{n=1}^{N} h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right) x_{n,j} x_{n,k}$$

The Hessian is a matrix of second derivatives, so the jk-th element of the Hessian is $\frac{\partial^2 J}{\partial \theta_j \partial \theta_k}$. Similarly, the jk-th element of $x_n x_n^T$ is $x_{n,j} x_{n,k}$, so the Hessian is the sum of the outer products of the gradient of the logistic function evaluated at each data point. Thus, the Hessian is given by

$$\boxed{\boldsymbol{H} = \sum_{n=1}^{N} h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\right)\boldsymbol{x}_n \boldsymbol{x}_n^T}$$

(c) **(5 pts)** Show that J is a convex function and therefore has no local minima other than the global one.
Hint: A function J is convex if its Hessian is positive semi-definite (PSD), written $\boldsymbol{H} \succeq 0$. A matrix is PSD if and only if

$$\boldsymbol{z}^T \boldsymbol{H} \boldsymbol{z} \equiv \sum_{j,k} z_j z_k H_{jk} \geq 0.$$

for all real vectors $\boldsymbol{z}$.

Solution:

The Hessian is given by

$$\begin{aligned}\boldsymbol{H} &= \sum_{n=1}^{N} h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)\left(1 - h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)\right)\boldsymbol{x}_n\boldsymbol{x}_n^T \\ &= \sum_{n=1}^{N} h'_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)\boldsymbol{x}_n\boldsymbol{x}_n^T.\end{aligned}$$

We can show that the Hessian is positive semi-definite by showing that the quadratic form

$$\begin{aligned}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} &= \boldsymbol{z}^T\left(\sum_{n=1}^{N} h'_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)\boldsymbol{x}_n\boldsymbol{x}_n^T\right)\boldsymbol{z} \\ &= \sum_{n=1}^{N} \boldsymbol{z}^T\left(h'_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)\boldsymbol{x}_n\boldsymbol{x}_n^T\right)\boldsymbol{z}\end{aligned}$$

is non-negative for all real vectors $\boldsymbol{z}$. Since $h'_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)$ is a scalar, we can pull it out of the quadratic form. Then by associativity of matrix multiplication, we have

$$\begin{aligned}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} &= \sum_{n=1}^{N} h'_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)\left(\boldsymbol{z}^T\boldsymbol{x}_n\right)\left(\boldsymbol{x}_n^T\boldsymbol{z}\right) \\ &= \sum_{n=1}^{N} h'_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)\left(\boldsymbol{x}_n^T\boldsymbol{z}\right)^T\left(\boldsymbol{x}_n^T\boldsymbol{z}\right) \\ &= \sum_{n=1}^{N} h'_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)\left(\boldsymbol{x}_n^T\boldsymbol{z}\right)^2\end{aligned}$$

Since $h'_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right)$ is a probability, it is non-negative. Further, because $\boldsymbol{x}_n$ and $\boldsymbol{z}$ have the same dimension, $\boldsymbol{x}_n^T\boldsymbol{z}$ is a scalar, so $\left(\boldsymbol{x}_n^T\boldsymbol{z}\right)^2$ is also non-negative. Thus, the quadratic form is non-negative, so the Hessian is positive semi-definite. Therefore, J is a convex function.

3 Maximum Likelihood Estimation [15 pts]

Suppose we observe the values of n independent random variables $X_1, \ldots, X_n$ drawn from the same Bernoulli distribution with parameter θ[1]. In other words, for each X_i, we know that

$$P(X_i = 1) = \theta \quad \text{and} \quad P(X_i = 0) = 1 - \theta.$$

Our goal is to estimate the value of θ from these observed values of X_1 through X_n.

For any hypothetical value $\hat{\theta}$, we can compute the probability of observing the outcome $X_1, \ldots, X_n$ if the true parameter value θ were equal to $\hat{\theta}$. This probability of the observed data is often called the *likelihood*, and the function $L(\theta)$ that maps each θ to the corresponding likelihood is called the *likelihood function*. A natural way to estimate the unknown parameter θ is to choose the θ that maximizes the likelihood function. Formally,

$$\hat{\theta}_{MLE} = \arg\max_{\theta} L(\theta).$$

(a) **(3 pts)** Write a formula for the likelihood function, $L(\theta) = P(X_1, \ldots, X_n; \theta)$. Your function should depend on the random variables $X_1, \ldots, X_n$ and the hypothetical parameter θ. Does the likelihood function depend on the order in which the random variables are observed?

Solution: Let $m = \sum_{i=1}^{n} X_i$ be the number of 1s in the data set. Because each event is independent, the probability of observing the data set is the product of the probabilities of observing each individual event. Thus, the likelihood function is given by

$$\boxed{L(\theta) = \theta^m (1-\theta)^{n-m}.}$$

Since multiplication is commutative, the likelihood function does not depend on the order in which the random variables are observed.

(b) **(6 pts)** Since the log function is increasing, the θ that maximizes the *log likelihood* $\ell(\theta) = \log(L(\theta))$ is the same as the θ that maximizes the likelihood. Find $\ell(\theta)$ and its first and second derivatives, and use these to find a closed-form formula for the MLE. For this part, assume we use the natural logarithm (i.e. logarithm base e).

Solution: The log likelihood function is given by

$$\begin{aligned} l(\theta) &= \log(L(\theta)) = \log(\theta^m (1-\theta)^{n-m}) \\ &= m \log(\theta) + (n-m) \log(1-\theta). \end{aligned}$$

The first derivative of the log likelihood function is

$$l'(\theta) = \frac{m}{\theta} - \frac{n-m}{1-\theta}.$$

[1] This is a common assumption for sampling data. So we will denote this assumption as iid, short for Independent and Identically Distributed, meaning that each random variable has the same distribution and is drawn independent of all the other random variables

To find the MLE, we set the first derivative to zero and solve for θ:

$$\begin{aligned} l'(\theta) &:= 0 \\ &= \frac{m}{\theta} - \frac{n-m}{1-\theta} \\ &= \frac{m(1-\theta) - \theta(n-m)}{\theta(1-\theta)} \\ &= \frac{m - m\theta - n\theta + m\theta}{\theta(1-\theta)} \\ &= \frac{m - n\theta}{\theta(1-\theta)} \end{aligned}$$

Thus, the critical points are $\theta = 0$, $\theta = 1$, and $\theta = \frac{m}{n}$. We can immediately classify $\theta = 0$ and $\theta = 1$ as minima because $L(0) = 0$ and $L(1) = 0$. To determine whether $\theta = \frac{m}{n}$ is a maximum or minimum, we can use the second derivative test. The second derivative of the log likelihood function is

$$l''(\theta) = -\frac{m}{\theta^2} - \frac{n-m}{(1-\theta)^2}.$$

Substituting $\theta = \frac{m}{n}$, we have

$$\begin{aligned} l''\left(\frac{m}{n}\right) &= -\frac{n^2}{m} - \frac{n^2}{n-m} \\ &= \frac{-n^2(n-m) - n^2 m}{m(n-m)} \\ &= \frac{-n^3}{m(n-m)}. \end{aligned}$$

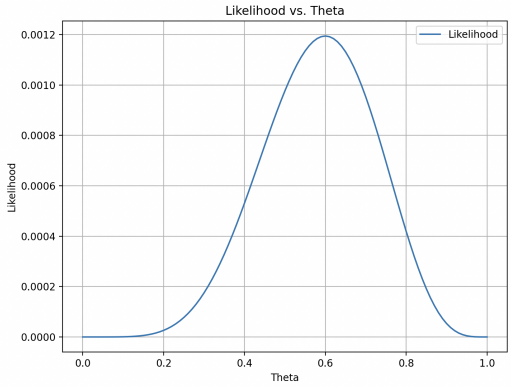
Since n, m, and $n-m$ are all positive, $l''\left(\frac{m}{n}\right) < 0$, so $\theta = \frac{m}{n}$ is a maximum. Thus, the MLE is given by $\boxed{\hat{\theta}_{MLE} = \frac{m}{n}}$.

(c) **(3 pts)** Suppose that $n = 10$ and the data set contains six 1s and four 0s. Write a short program `likelihood.py` that plots the likelihood function of this data for each value of θ in $\{0, 0.01, 0.02, \ldots, 1.0\}$ (use `np.linspace(...)` to generate this spacing). For the plot, the x-axis should be θ and the y-axis $L(\theta)$. Scale your y-axis so that you can see some variation in its value. Include the plot in your writeup (there is no need to submit your code). Estimate $\hat{\theta}_{MLE}$ by marking on the x-axis the value of θ that maximizes the likelihood. Does the answer agree with the closed form answer?

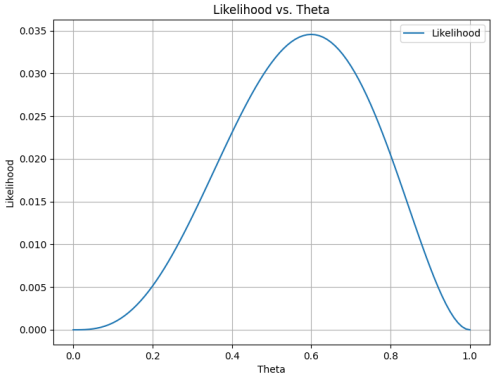
Solution: The likelihood function is given by $L(\theta) = \theta^6(1-\theta)^4$. The plot of the likelihood function is shown on the page below. The maximum likelihood estimate is $\hat{\theta}_{MLE} = \frac{6}{10} = 0.6$, which agrees with the closed form answer.

(d) **(3 pts)** Create three more likelihood plots: one where $n = 5$ and the data set contains three 1s and two 0s; one where $n = 100$ and the data set contains sixty 1s and forty 0s; and one where $n = 10$ and there are five 1s and five 0s. Include these plots in your writeup, and describe how the likelihood functions and maximum likelihood estimates compare for the different data sets.

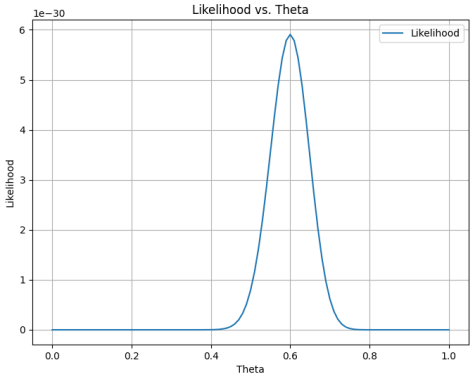
Solution: The likelihood functions and maximum likelihood estimates for the different data sets are shown below. As the number of data points increases, the likelihood function becomes more peaked around the maximum likelihood estimate. The maximum likelihood estimate is always equal to the proportion of 1s in the data set, which is consistent with the closed form answer.



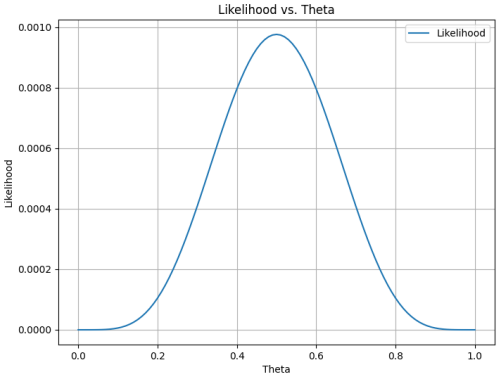
(a) Likelihood function for $n = 10$, $m = 6$



(b) Likelihood function for $n = 5$, $m = 3$



(c) Likelihood function for $n = 100$, $m = 60$



(d) Likelihood function for $n = 10$, $m = 5$

Figure 1: Likelihood functions for different data sets

4 Implementation: Polynomial Regression [20 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: `CS146_Winter2024_PS2.ipynb`
- data: `regression_train.csv`, `regression_test.csv`

Similar to *PS1*, copy the colab notebook to your drive and make the changes to the notebook. Mount the drive appropriately and copy the data to your drive to access via colab.

The notebook has marked blocks where you need to code.

$$\#\#\# \text{ ========= } TODO : START \text{ ========= } \#\#\#$$

$$\#\#\# \text{ ========= } TODO : END \text{ ========== } \#\#\#$$

Note: For parts b, c, g, and h (and only these parts), you are expected to copy-paste/screenshot your code snippet as a part of the solution in the submission pdf. Tip: If you are using LaTeX, check out the Minted package (example) for code highlighting.

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.[2] 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[3]. 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 $1 \times N$.

[2] Try out `SciPy`'s tutorial (https://numpy.org/doc/stable/user/quickstart.html), or use your favorite search engine to find an alternative. Those familiar with Matlab may find the "Numpy for Matlab Users" documentation (https://numpy.org/doc/stable/user/numpy-for-matlab-users.html) more helpful.

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

Visualization [1 pts]

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) **(1 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: The training and test data are shown in the scatter plots below. It appears like a linear model could be a good fit for the data, but there are some outliers that may make the model less effective. A cubic model may be a better fit for the data.

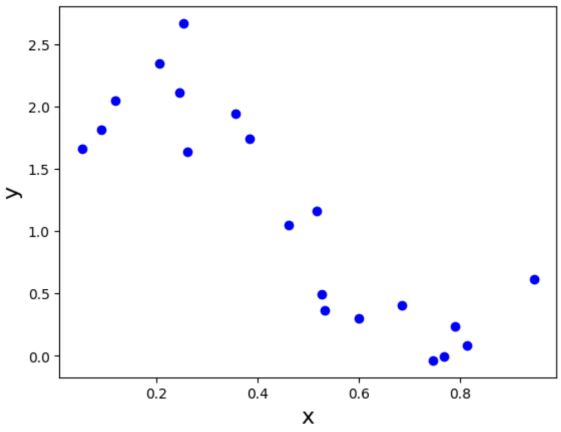


Figure 2: Training data

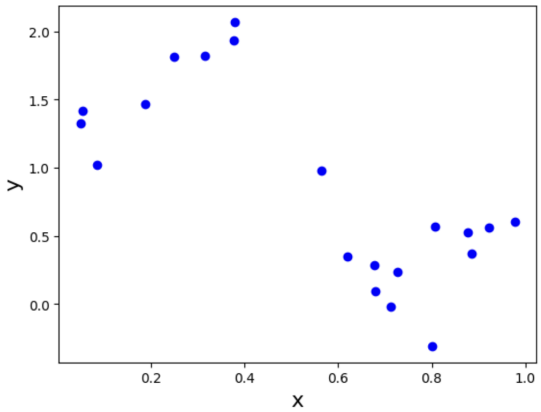


Figure 3: Test data

Linear Regression [12 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:

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

The Colab notebook 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 is $\left(1, x_{n,1}, x_{n,1}^2, \ldots, x_{n,1}^m\right)^T$. Setting $m = 1$ instantiates an object where the feature vector for instance n is $\left(1, x_{n,1}\right)^T$.

(b) **(1 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. $x_{n,0} = 1$. 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. Copy-paste/screenshot your code snippet.
Solution:

```
intercept = np.ones((n, 1))
Phi = np.hstack((intercept, X))
```

(c) **(1 pts)** Before tackling the harder problem of training the regression model, complete `PolynomialRegression.predict(...)` to predict $\boldsymbol{y}$ from $\boldsymbol{X}$ and $\boldsymbol{\theta}$. Copy-paste/screenshot your code snippet.
Solution:

```
y = X @ self.coef_
```

(d) **(5 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 $J(\boldsymbol{\theta})$.

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

In gradient descent, each iteration performs the update

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

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 $J(\boldsymbol{\theta})$.

- 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 $J(\boldsymbol{\theta})$.
  Solution:

```
diffs = (self.predict(X) - y) ** 2
cost = np.sum(diffs, axis=0)
```

- 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 earlier if the value of the objective function is unchanged across consecutive iterations.
- We will use a fixed learning rate.

```
error = (X @ self.coef_)-y
grad = 2 * (X.T @ error)
self.coef_ = self.coef_ - eta * grad
y_pred = X @ self.coef_
```

- Experiment with different values of learning rate $\eta = 10^{-4}, 10^{-3}, 10^{-2}, 0.1$, 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: The coefficients, number of iterations until convergence, and the final value of the objective function for different learning rates are shown in the table below. The algorithm converges fastest for a learning rate of 0.01 but has the same objective value as a learning rate of 0.001. The algorithm diverges for a learning rate of 0.1.

Learning Rate	Number of Iterations	Coefficients	Train Cost	Test Cost
0.0001	10000	[2.270, -2.461]	4.086	5.841
0.001	7020	[2.446, -2.816]	3.913	7.047
0.01	764	[2.446, -2.816]	3.913	7.047
0.1	10000	[nan, nan]	nan	nan

(e) **(4 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.

- Implement the closed-form solution `PolynomialRegression.fit(...)`.

```
self.coef_ = np.linalg.pinv(X.T @ X) @ X.T @ y
```

- What is the closed-form solution coefficients? How do the coefficients and the cost compare to those obtained by GD? Use the 'time' module to compare its runtime to GD.
  Solution: The coefficients and the cost for the closed-form solution are the same as those obtained by GD with an optimized learning rate. The runtime for the closed-form solution is much faster than GD.

Algorithm	Time Elapsed	Coefficients	Train Cost	Test Cost
Gradient Descent ($\epsilon = 0.01$)	0.01757 s	[2.446 -2.816]	3.912	7.047
Closed-Form	0.00029 s	[2.446 -2.816]	3.912	7.047

(f) **(1 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 many iterations does it take the algorithm to converge with your proposed learning rate?

Solution: The algorithm converges with the proposed learning rate after 1718 iterations.

Polynomial Regression [7 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) **(1 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. Copy-paste/screenshot your code snippet.

Solution:

```
Phi = np.ones((n, 1))
for i in range(1, m + 1):
  Phi = np.hstack((Phi, X**i))
```

(h) **(2 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.[4]

Why do you think we might prefer RMSE as a metric over $J(\boldsymbol{\theta})$?

Implement `PolynomialRegression.rms_error(...)`. Copy-paste/screenshot your code snippet.

Solution: The RMSE measures the average magnitude of the errors in the model, so it is a better measure of the model's performance than the sum of the squared errors $J(\boldsymbol{\theta})$, which scales with the number of data points. The RMSE is also scale-invariant, so it can be used to compare models with different units of measurement.

[4] 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, $k = m+1$.

```
N, d = X.shape
        error = np.sqrt(self.cost(X,y) / N)
```

(i) **(4 pts)** For $m = 0, \ldots, 10$ (where m is the degree of the polynomial), 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: The RMSE for the training data decreases as the polynomial degree increases, while the RMSE for the test data decreases initially and then increases. The best-fit polynomial degree is $m = 5$, which has the lowest RMSE for the test data. The RMSE for the test data increases for higher polynomial degrees while training error stays low, which is evidence of overfitting.

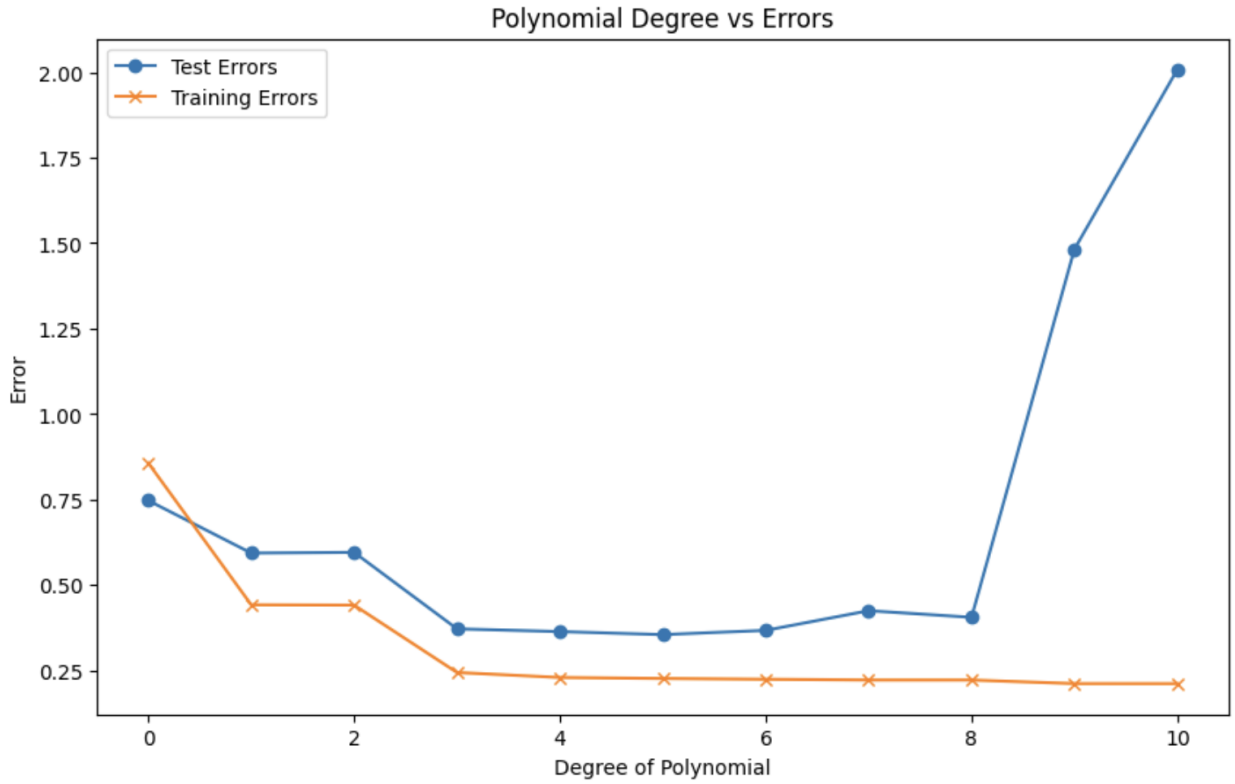


Figure 4: Train and test RMSE for different polynomial degrees