CS 446 / ECE 449 — Homework 1

scs13

Version 1.0

Instructions.

- Homework is due **Tuesday, February 8, at noon CST**; no late homework accepted.
- Everyone must submit individually at gradescope under `hw1` and `hw1code`.
- The "written" submission at `hw1` **must be typed**, and submitted in any format gradescope accepts (to be safe, submit a PDF). You may use LaTeX, Markdown, Google Docs, MS Word, whatever you like; but it must be typed!
- When submitting at `hw1`, gradescope will ask you to mark out boxes around each of your answers; please do this precisely!
- Please make sure your NetID is clear and large on the first page of the homework.
- Your solution **must** be written in your own words. Please see the course webpage for full academic integrity information. Briefly, you may have high-level discussions with at most 3 classmates, whose NetIDs you should place on the first page of your solutions, and you should cite any external reference you use; despite all this, your solution must be written in your own words.
- We reserve the right to reduce the auto-graded score for `hw1code` if we detect funny business (e.g., your solution lacks any algorithm and hard-codes answers you obtained from someone else, or simply via trial-and-error with the autograder).
- Coding problems come with suggested "library routines"; we include these to reduce your time fishing around APIs, but you are free to use other APIs.
- When submitting to `hw1code`, only upload the two python files `hw1.py` and `hw1_utils.py`. Don't upload a zip file or additional files.

Version history.

1. Initial version.

1. Linear Regression/SVD.

Throughout this problem let $\boldsymbol{X}$ be the $n \times d$ matrix with the feature vectors $(\boldsymbol{x}_i)_{i=1}^n$ as its rows. Suppose we have the singular value decomposition $\boldsymbol{X} = \sum_{i=1}^r s_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$.

(a) Let the training examples $(\boldsymbol{x}_i)_{i=1}^n$ be the standard basis vectors $\boldsymbol{e}_i$ of $\mathbb{R}^d$ with each $\boldsymbol{e}_i$ repeated $n_i > 0$ times having labels $\left(y_{i_j}\right)_{j=1}^{n_i}$. That is, our training set is:

$$\bigcup_{i=1}^{d} \left\{ \left(\boldsymbol{e}_i, y_{i_j}\right) \right\}_{j=1}^{n_i},$$

where $\sum_{i=1}^d n_i = n$. Show that for a vector $\boldsymbol{w}$ that minimizes the empirical risk, the components w_i of $\boldsymbol{w}$ are the averages of the labels $\left(y_{i_j}\right)_{j=1}^{n_i}$: $w_i = \frac{1}{n_i}\sum_{j=1}^{n_i} y_{i_j}$.

Hint: Write out the expression for the empirical risk with the squared loss and set the gradient equal to zero.

Remark: This gives some intuition as to why "regression" originally meant "regression towards the mean."

(b) Returning to a general matrix $\boldsymbol{X}$, show that if the label vector $\boldsymbol{y}$ is a linear combination of the $\{\boldsymbol{u}_i\}_{i=1}^r$ then there exists a $\boldsymbol{w}$ for which the empirical risk is zero (meaning $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$).

Hint: Either consider the range of $\boldsymbol{X}$ and use the SVD, or compute the empirical risk explicitly with $\boldsymbol{y} = \sum_{i=1}^r a_i \boldsymbol{u}_i$ for some constants a_i and $\hat{\boldsymbol{w}}_{\text{ols}} = \boldsymbol{X}^+\boldsymbol{y}$.

Remark: It's also not hard to show that if $\boldsymbol{y}$ is not a linear combination of the $\{\boldsymbol{u}_i\}_{i=1}^r$, then the empirical risk must be nonzero.

(c) Show that $\boldsymbol{X}^\top\boldsymbol{X}$ is invertible if and only if $(\boldsymbol{x}_i)_{i=1}^n$ spans $\mathbb{R}^d$.

Hint: Recall that the squares of the singular values of $\boldsymbol{X}$ are eigenvalues of $\boldsymbol{X}^\top\boldsymbol{X}$.

Remark: This characterizes when linear regression has a unique solution due to the normal equation (note that we always have at least one solution obtained by the pseudoinverse). We would not have had a unique solution for part (a) if we had an $n_i = 0$.

(d) Provide a matrix $\boldsymbol{X}$ such that $\boldsymbol{X}^\top\boldsymbol{X}$ is invertible and $\boldsymbol{X}\boldsymbol{X}^\top$ is not. Include a formal verification of this for full points.

Hint: Use part (c). It may be helpful to think about conditions under which a matrix is not invertible.

Solution.

(a) The equation for empirical risk as given in the question is: $R_{log}(w) = \frac{1}{n}\sum_{i=1}^n (w^T x_i - y_i)x_i = 0$

To find the w that minimizes the risk, gradient descent principles are applied. First, the gradient of the above risk function is calculated by taking the differential of the above equation with respect to w:
$\nabla \hat{R}_{log}(w) = \frac{1}{n}\sum_{i=1}^n (w^T x_i - y_i)x_i$

The obtained equation is equated to zero:
$\nabla \hat{R}_{log}(w) = \frac{1}{n}\sum_{i=1}^n (w^T x_i - y_i)x_i = 0$

The multiplication terms can be expanded to get:
$\sum_{i=1}^n (w^T x_i)x_i - \sum_{i=1}^n (y_i)x_i = 0$

Upon rearrangement of terms:
$\sum_{i=1}^n (w^T x_i)x_i = \sum_{i=1}^n (y_i)x_i$

The above matrix form of w can be expanded to a scalar form by summation over i (from 1 to d) and j (from 1 to n_)

$\sum_{i=1}^d \sum_{j=1}^{n_i} (w_i x_i) = \sum_{i=1}^d (x_i \sum_{j=1}^{n_i} (y_{i_j}))$

The terms inside the summation must equate. Thus:
$\sum_{j=1}^{n_i}(w_i x_i) = x_i \sum_{j=1}^{n_i}(y_{i_j})$

The independence of x_i from x_j ensures the cancellation on both sides, simplifying the process:

$\sum_{j=1}^{n_i}(w_i) = \sum_{j=1}^{n_i}(y_{i_j})$

Thus,

$w_i = \frac{1}{n_i}\sum_{j=1}^{n_i}(y_{i_j})$

(b) The goal here is to ensure that the predicted outcome is as close to the expected outcome as possible. For this, we can start by defining y: As a linear combination of $\{\boldsymbol{u}_i\}_{i=1}^{r}$,
y = $\boldsymbol{y} = \sum_{i=1}^{r} a_i \boldsymbol{u}_i$ The optimal value of w is given by:
$\hat{\boldsymbol{w}}_{\text{ols}} = \boldsymbol{X}^{+}\boldsymbol{y}$
By applying SVD to this, we get:
$\hat{\boldsymbol{w}}_{\text{ols}} = \sum_{i=1}^{r} \frac{1}{s_i}\boldsymbol{v}_i\boldsymbol{u}_i^{\top}$ ă$_i\boldsymbol{u}_i$

The constant terms of a can be pulled out together. The unitary matrices u_i and $\boldsymbol{u}_i^{\top}$ can be grouped together to give out either 0 or 1 (due to their orthonormal property).
Now, the prediction value Xw can be calculated:
$\boldsymbol{X}\boldsymbol{w} = \sum_{i=1}^{r} s_i\boldsymbol{u}_i\boldsymbol{v}_i^{\top}\frac{1}{s_i}a_i\boldsymbol{v}_i$.
Pulling together all the constant terms, we have:
$\boldsymbol{X}\boldsymbol{w} = \sum_{i=1}^{r} s_i a_i \boldsymbol{u}_i$
Thus, it can be concluded that under these conditions, empirical risk is zero.

(c) The $(\boldsymbol{x}_i)_{i=1}^{n}$ basis vectors are said to span across the real numbers space $\mathbb{R}^d$. Intuitively, there exist eigen value(s) which are zeros. This is due to the existence of null basis vectors. Another way to check for this is the lack of full rank of the matrix $\boldsymbol{X}$. The SVD of X is given by:
$\sum_{i=1}^{r} s_i\boldsymbol{u}_i\boldsymbol{v}_i^{\top}$.
The transpose of this given by:
$\boldsymbol{X}^{\top} = \sum_{i=1}^{r} s_i\boldsymbol{v}_i\boldsymbol{u}_i^{\top}$.
When $\boldsymbol{X}^{\top}\boldsymbol{X}$ is calculated, we get:
$\boldsymbol{X}^{\top}\boldsymbol{X} = \sum_{i=1}^{r} s_i^2\boldsymbol{v}_i\boldsymbol{u}_i^{\top}\boldsymbol{u}_i\boldsymbol{v}_i^{\top}$.
The values of s can be pulled out because it is a scalar value. The unitary vectors $\boldsymbol{u}_i^{\top}\boldsymbol{u}_i$ result in either 0 or 1 due to their orthonormal property. Thus, the equation can be written as:
$\boldsymbol{X}^{\top}\boldsymbol{X} = \sum_{i=1}^{r} s_i^2\boldsymbol{v}_i\boldsymbol{v}_i^{\top}$ The square of singular values of X that we obtain here form the eigen values of $\boldsymbol{X}^{\top}\boldsymbol{X}$.

(d) One of the simplest counter examples that can prove the above statement is: X = [2 2]. In fact, any repeated pair of numbers in X ([1 1],[3 3],[100 100]..) satisfy the above condition.
Proof: The dimension of matrix X is given by nxd. The dimension of $\boldsymbol{X}^{\top}$ is dxn. The dimension of $\boldsymbol{X}^{\top}\boldsymbol{X}$ is given by nxn and the dimension of $\boldsymbol{X}\boldsymbol{X}^{\top}$ is given by dxd.
Consider the case when n $\neq$ d:
Either n d or d n. The smallest of the two will be the rank. Suppose n is smaller so it is the rank. The matrix with nxn will then have full rank, all x$_i$ spanning over the real vales. However, the bigger matrix of dimension dxd will have d-n null spaces, making it non-invertible.

2. Logistic Loss.

Throughout this problem, let $\ell_{\log}(z) := \ln(1 + \exp(-z))$ denote the logistic loss. Recall the empirical risk $\widehat{\mathcal{R}}$ for logistic regression (as presented in lecture 3):

$$\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n} \sum_{k=1}^{n} \ln(1 + \exp(-y_k \boldsymbol{w}^\top \boldsymbol{x}_k)).$$

As in lecture, the corresponding empirical risk $\widehat{\mathcal{R}}_{\log}$ can not in general be minimized by solving for $\boldsymbol{w}$ in $\nabla\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = 0$, but here we will consider a special case where it is possible. Specifically, consider the setting of part (a) of the previous problem: the training examples $(\boldsymbol{x}_i)_{i=1}^{n}$ are the standard basis vectors $\boldsymbol{e}_i$ of $\mathbb{R}^d$, each repeated respectively $n_i > 0$ times; one difference is that now $y_{i_j} \in \{+1, -1\}$, rather than being arbitrary elements of $\mathbb{R}$.

Crucially, throughout this problem, for each i suppose that $n_i \geq 2$, and there exist at least one positive and one negative label.

(a) Recall that the logistic loss suggests a conditional probability model that the label of a given example is $+1$: for any $\boldsymbol{w}$ and any $\boldsymbol{x}$, the corresponding conditional model $\hat{p}_{\boldsymbol{w}}$ is

$$\hat{p}_{\boldsymbol{w}}(\boldsymbol{x}) := \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})}.$$

Show that in the above setting, consider a given i,

$$\hat{p}_{\hat{\boldsymbol{w}}}(\boldsymbol{e}_i) := \frac{\left|\{j : y_{i_j} = +1\}\right|}{n_i},$$

where $\hat{\boldsymbol{w}}$ denotes the optimal solution to $\widehat{\mathcal{R}}_{\log}$ (which exists in this setting).

For full points, you must include and justify a derivation.

(b) Now suppose that for some k, $n_k \to \infty$; specifically, there is only one positive pair $(\boldsymbol{e}_k, +1)$, and $n_k - 1$ negative pairs $(\boldsymbol{e}_k, -1)$. correspondingly, let $\hat{\boldsymbol{w}}_n$ denote the optimal solution for each sample size.

Determine and formally prove the limiting behavior of $\|\hat{\boldsymbol{w}}_n\|$ and $\hat{p}_{\hat{\boldsymbol{w}}_n}(\boldsymbol{e}_k)$.

(c) Now consider the case of a fixed finite n, but let's violate our assumption above regarding $n_i \geq 2$: specifically, suppose there exists $\boldsymbol{e}_i$ with $n_i = 1$.

We can no longer compute an optimal $\hat{\boldsymbol{w}} \in \mathbb{R}^d$ (why not?), but based on the preceding reasoning, we can make a guess for $\hat{p}_{\hat{\boldsymbol{w}}}$.

What do you think it is? And, as a consequence, do you think $\hat{p}_{\hat{\boldsymbol{w}}}$ represents a reasonable notion of "prediction confidence"?

Grading note: Any answer to this problem which consists of at least one complete sentence will receive full credit. That said, as the use of confidences in this form is standard in machine learning, please consider spending a bit of time.

Solution. (a) The empirical risk given for logistic regression is:

$$\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n} \sum_{k=1}^{n} \ln(1 + \exp(-y_k \boldsymbol{w}^\top \boldsymbol{x}_k)).$$

To find the optimal w ($\hat{w}$) can be obtained by gradient descent method, where we can differentiate the loss function (empirical risk) with respect to w and then equate it to 0:
$\nabla\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{(1+\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))} * (\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) * (-y_i \boldsymbol{x}_i)$

$\nabla\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{(1+\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))} * (\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) * (-y_i \boldsymbol{x}_i) = 0$

Algebraic manipulations of the above equation result in:

$\nabla\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \frac{y_i \boldsymbol{x}_i}{(1+\exp(y_i \boldsymbol{w}^\top \boldsymbol{x}_i))}$

This can be considered as the updating step:
$w' \leftarrow w - \eta(\frac{-1}{n}\sum_{i=1}^{n} \frac{y_i x_i}{1+\exp{(y_i w^T x_i)}})$

The values of y are +1 or -1. Thus, the above summation can be split into two - one containing all the +1 and the other, -1. This splitting also occurs for the summation terms. The conditional probability enables the splitting of both. Now, the summations go from i = 1 to n and j = 1 to d.
$\sum_{i=1}^{d}(\frac{|j:y_{i_j}=+1|}{1+\exp{(y_i w^T x_i}} + \sum_{i=1}^{d} \frac{|j:y_{i_j}=-1|}{1+\exp{(y_i w^T x_i)}} = 0$
This can be rearranged to obtain:

$\frac{|j:y_{i_j}=+1|}{1+\exp(w_i)} = \frac{|j:y_{i_j}=-1|}{1+\exp(-w_i)}$

Let us consider the count of p(e_i) to be equal to some number m that lies between 1 to n: $(\frac{|j:y_{i_j}=+1|}{1+\exp(w_i)} = m)$

$\frac{m}{1+\exp(w_i)} = \frac{n_i - m}{1+\exp(-w_i)}$

$\frac{n_i - m}{m} = \frac{1+\exp(-w_i)}{1+\exp(w_i)}$

This can be simplified to obtain: $w_i = -\ln(\frac{m}{n_i - m})$

$\hat{p}_{\hat{w}}(e_i) = \frac{1}{1+\exp(-w_i)} = \frac{1}{1+\exp(\ln(\frac{n_i - m}{m}))}$

$\frac{1}{1+\frac{n_i - m}{m}}$

$\Rightarrow \frac{m}{n_i}$

(b) There is only one pair of +1, e$_k$. This means that there are ∞-1 pairs of (e$_k$,-1). The analytical value of w from part (a) dictates that the denominator is dependent on n$_k$. Since this tends to ∞, the $\hat{p}_{\hat{\boldsymbol{w}}}$ value tends to 0. The value of w (2-norm) tends to 0 as well.s

(c) In the case of fixed finite n, specifically when n$_i$ = 1, then the output pair can either be (e$_k$,-1) or (e$_k$,+1). The outcome of the probability is then either 1 or 0. This lack existence of both +1 and -1 pairings occurring together makes this computation infeasible.

3. Linear Regression.

Recall that the empirical risk in the linear regression method is defined as $\widehat{\mathcal{R}}(\boldsymbol{w}) := \frac{1}{2n}\sum_{i=1}^{n}(\boldsymbol{w}^{\top}\boldsymbol{x}_i - y_i)^2$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is a data point and y_i is an associated label.

(a) Implement linear regression using gradient descent in the `linear_gd(X, Y, lrate, num_iter)` function of `hw1.py`. You are given as input a training set `X` as an $n \times d$ tensor, training labels `Y` as an $n \times 1$ tensor, a learning rate `lrate`, and the number of iterations of gradient descent to run `num_iter`. Using gradient descent, find parameters $\boldsymbol{w}$ that minimize the empirical risk $\widehat{\mathcal{R}}(\boldsymbol{w})$. Use $\boldsymbol{w} = 0$ as your initial parameters, and return your final w as output. Prepend a column of ones to `X` in order to accommodate a bias term in $\boldsymbol{w}$.

Library routines: `torch.matmul (@), torch.tensor.shape, torch.tensor.t, torch.cat, torch.ones, torch.zeros, torch.reshape`.

(b) Implement linear regression by using the pseudoinverse to solve for w in the `linear_normal(X,Y)` function of `hw1.py`. You are given a training set `X` as an $n \times d$ tensor and training labels `Y` as an $n \times 1$ tensor. Return your parameters w as output. As before, make sure to accommodate a bias term by prepending ones to the training examples `X`.

Library routines: `torch.matmul (@), torch.cat, torch.ones, torch.pinverse`.

(c) Implement the `plot_linear()` function in `hw1.py`. Use the provided function `hw1_utils.load_reg_data()` to generate a training set `X` and training labels `Y`. Plot the curve generated by `linear_normal()` along with the points from the data set. Return the plot as output. Include the plot in your written submission.

Library routines: `torch.matmul (@), torch.cat, torch.ones, plt.plot, plt.scatter, plt.show, plt.gcf` where `plt` refers to the `matplotlib.pyplot` library.

Solution.

(a) The linear_gd function is written to take in the values of X, Y, learning rate and number of iterations and output the weight(s) w. The input X is pre-pended with a column of ones by using the torch.ones and the torch.cat routines. The shape of this X gives the number of samples and the number of features. The weights w are initialized to zeros based on the number of features in X. The training involves predicting the values of y, calculating the loss and performing gradient descent to obtain the step update. This is iteratively performed num_iter times. The prediction of y is done by the forward function. This function returns X*w. The loss function is then called to obtain the empirical risk value using the formula: $\widehat{\mathcal{R}}(\boldsymbol{w}) := \frac{1}{2n}\sum_{i=1}^{n}(\boldsymbol{w}^{\top}\boldsymbol{x}_i - y_i)^2$. The gradient descent value is then calculated using the update step: w' = w - $\eta\nabla$R. The code and output have been tested on autograder.

(b) The plot_linear function takes in the training set X and training labels Y as inputs and returns w as output. The calculation is done similar to the one in 3(a). The X variable is pre-pended with a column of ones and the w is calculated as w = X^{+}y, where X^{+} is the pseudo inverse of X. The returned w is the required answer. The code for this has been implemented and tested on auto grader.

(c) The plot_linear function has X and Y values taken from load_reg_data function from utils. The linear_normal function is called by weight variable w to calculate the weights as mentioned in 3(b). X is pre-pended with a column of ones. Finally, we obtain the plots of X with Y and X with predicted Y.

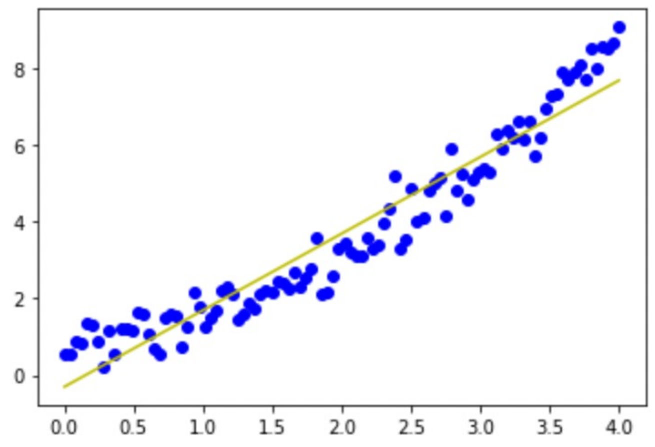


Figure 1: Linear Regression

4. Polynomial Regression.

In Problem 3 you constructed a linear model $\boldsymbol{w}^\top \boldsymbol{x} = \sum_{i=1}^{d} x_i w_i$. In this problem you will use the same setup as in the previous problem, but enhance your linear model by doing a quadratic expansion of the features. Namely, you will construct a new linear model $f_{\boldsymbol{w}}$ with parameters

$$(w_0, w_{01}, \ldots, w_{0d}, w_{11}, w_{12}, \ldots, w_{1d}, w_{22}, w_{23}, \ldots, w_{2d}, \ldots, w_{dd})^\top,$$

defined by

$$f_{\boldsymbol{w}}(x) = \boldsymbol{w}^\top \phi(\boldsymbol{x}) = w_0 + \sum_{i=1}^{d} w_{0i} x_i + \sum_{i \leq j}^{d} w_{ij} x_i x_j.$$

Warning: If the computational complexity of your implementation is high, it may crash the autograder (try to optimize your algorithm if it does)!

(a) Given a 3-dimensional feature vector $\boldsymbol{x} = (x_1, x_2, x_3)$, completely write out the quadratic expanded feature vector $\phi(\boldsymbol{x})$.

(b) Implement the `poly_gd()` function in `hw1.py`. The input is in the same format as it was in Problem 3. Implement gradient descent on this training set with $\boldsymbol{w}$ initialized to 0. Return $\boldsymbol{w}$ as the output with terms in this exact order: bias, linear, then quadratic. For example, if $d = 3$ then you would return $(w_0, w_{01}, w_{02}, w_{03}, w_{11}, w_{12}, w_{13}, w_{22}, w_{23}, w_{33})$.

Library routines: `torch.cat, torch.ones, torch.zeros, torch.stack.`

Hint: You will want to prepend a column of ones to `X`, and append to `X` the squared features in the specified order. You can generate the squared features in the correct order (This is important! The order of the polynomial features matters for your answer to match the correct answer on GradeScope. Check the order in the problem definition above.) using a nested for loop. We don't want duplicates (e.g., $x_0 x_1$ and $x_1 x_0$ should not both be included; we should only include $x_0 x_1$ in the quadratic case).

(c) Implement the `poly_normal` function in `hw1.py`. You are given the same data set as from part (b), but this time determine w by using the pseudoinverse. Return $\boldsymbol{w}$ in the same order as in part (b).

Library routines: `torch.pinverse.`

Hint: You will still need to transform the matrix `X` in the same way as in part (b).

(d) Implement the `plot_poly()` function in `hw1.py`. Use the provided function `hw1_utils.load_reg_data()` to generate a training set `X` and training labels `Y`. Plot the curve generated by `poly_normal()` along with the points from the data set. Return the plot as output and include it in your written submission. Compare and contrast this plot with the plot from Problem 3. Which model appears to approximate the data better? Justify your answer.

Library routines: `plt.plot, plt.scatter, plt.show, plt.gcf.`

(e) The Minsky-Papert XOR problem is a classification problem with data set:

$$X = \{(-1, +1), (+1, -1), (-1, -1), (+1, +1)\}$$

where the label for a given point (x_1, x_2) is given by its product $x_1 x_2$. For example, the point $(-1, +1)$ would be given label $y = (-1)(1) = -1$. Implement the `poly_xor()` function in `hw1.py`. In this function you will load the XOR data set by calling the `hw1_utils.load_xor_data()` function, and then apply the `linear_normal()` and `poly_normal()` functions to generate predictions for the XOR points. Include a plot of contour lines that show how each model classifies points in your written submission. Return the predictions for both the linear model and the polynomial model and use `contour_plot()` in `hw1_utils.py` to help with the plot. Do both models correctly classify all points? (Note that red corresponds to larger values and blue to smaller values when using `contour_plot` with the "coolwarm" colormap).

Hint: A "Contour plot" is a way to represent a 3-dimensional surface in a 2-D figure. In this example, the data points are pined to the figure with their features (x_1, x_2) as the coordinates in 2-D space (e.g., x and y axis); the third dimension (e.g., the predictions of the data points) is labeled on the points in the figure. The lines or curves that link the grid points with the same predictions together are called the "contours". See `contour_plot()` in `hw1_utils.py` for details.

Solution.

(a) If the vector $\boldsymbol{x} = (x_1, x_2, x_3)$, then the quadratic expanded feature vector $\phi(\boldsymbol{x})$ is given by - $x_1 + x_2 + x_3 + x_1^2 + x_2^2 + x_3^2 + x_1x_2 + x_1x_3 + x_2x_3$.

(b) The *poly_gd* function results in the necessary polynomial response of x and y by making use of the loss function and extending it to include polynomial terms. The non-quadratic terms can be calculated as done in question 3. The quadratic terms are a set of (i,j) combinations that range from 0 to d. These combinations are kept track of in the list combx. w is initialized to 0 and the training takes place in a similar manner as performed in *linear_gd*. The final result is w obtained in terms of w_0, $w_{01}, \ldots, w_{0d}, w_{11}, w_{12}, \ldots, w_{1d}, w_{22}, w_{23}, \ldots, w_{2d}, \ldots, w_{dd}$.

(c) The poly_normal function utilizes the same concepts as used in poly_gd. The expanded model consists of linear and quadratic terms constructed by using the values of X (pre-pended with a column of ones). The weight parameter w is given by X_+y. This is implemented using torch.pinverse and w is returned as the final output.

(d) The plot for poly_normal is as shown below: The polynomial regression curve is better than linear.

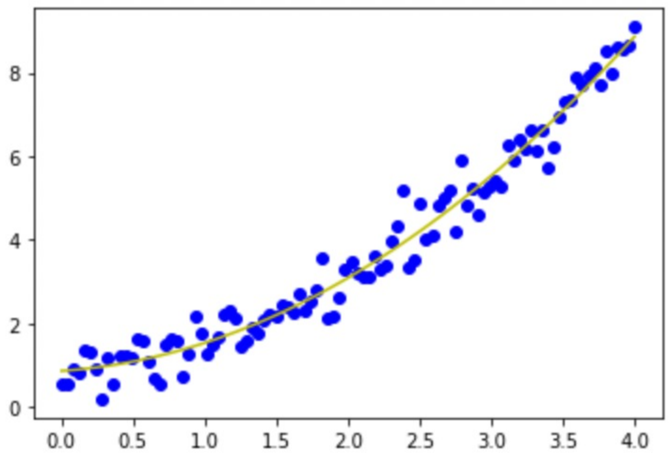
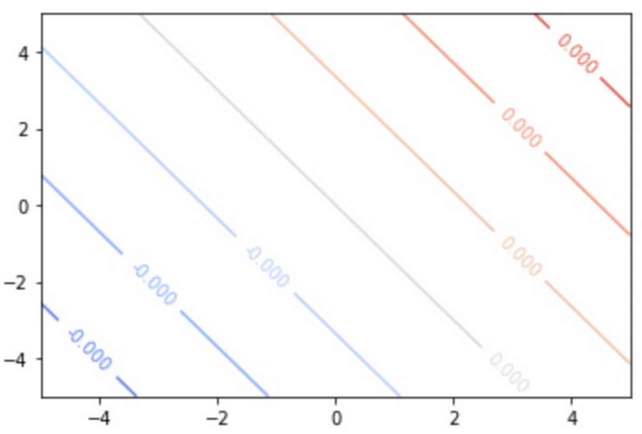


Figure 2: Polynomial Regression

It fits the data better by providing a better relationship between X and Y.

(e) Using the poly_xor function, the contour plot is as shown below. They do seem to classify data correctly.



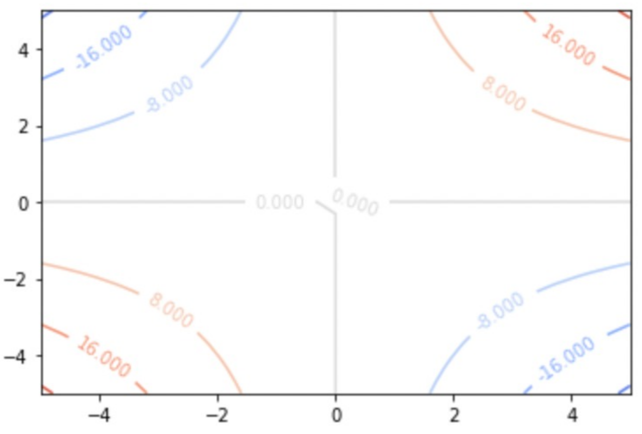


Figure 3: Contour of linear and polynomial normal

5. Logistic Regression.

Recall the empirical risk $\widehat{\mathcal{R}}$ for logistic regression (as presented in lecture 3):

$$\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)).$$

Here you will minimize this risk using gradient descent.

(a) In your written submission, derive the gradient descent update rule for this empirical risk by taking the gradient. Write your answer in terms of the learning rate η, previous parameters $\boldsymbol{w}$, new parameters $\boldsymbol{w}'$, number of examples n, and training examples $\boldsymbol{x}_i$. Show all of your steps.

(b) Implement the `logistic()` function in `hw1.py`. You are given as input a training set `X`, training labels `Y`, a learning rate `lrate`, and number of gradient updates `num_iter`. Implement gradient descent to find parameters $\boldsymbol{w}$ that minimize the empirical risk $\widehat{\mathcal{R}}_{\log}(\boldsymbol{w})$. Perform gradient descent for `num_iter` updates with a learning rate of `lrate`, initializing $\boldsymbol{w} = 0$ and returning $\boldsymbol{w}$ as output. Don't forget to prepend `X` with a column of ones.

Library routines: `torch.matmul (@), torch.tensor.t, torch.exp.`

(c) Implement the `logistic_vs_ols()` function in `hw1.py`. Use `hw1_utils.load_logistic_data()` to generate a training set `X` and training labels `Y`. Run `logistic(X,Y)` from part (b) taking `X` and `Y` as input to obtain parameters $\boldsymbol{w}$ (use the defaults for `num_iter` and `lrate`). Also run `linear_gd(X,Y)` from Problem 3 to obtain parameters $\boldsymbol{w}$. Plot the decision boundaries for your logistic regression and least squares models along with the data `X`. Which model appears to classify the data better? Explain why you believe your choice is the better classifier for this problem.

Library routines: `torch.linspace, plt.scatter, plt.plot, plt.show, plt.gcf.`

Hints:

- The positive and negative points are guaranteed to be linearly separable (though an algorithm may or may not find the optimal line to separate them).
- The "decision boundary" in the problem description refers to the set of points $\boldsymbol{x}$ such that $\boldsymbol{w}^\top \boldsymbol{x} = 0$ for the chosen predictor. In this case, it suffices to plot the corresponding line.
- In order to make the two models significantly different, we recommend that you train the logistic regression with a large `num_iter` (e.g., 1,000,000 or even larger).

Solution.

(a) The gradient descent update can be calculated by performing differentiation on the risk factor and set it to zero. The risk equation given by: $\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))$ is differentiated with respect to $\boldsymbol{w}$ and the result is:
$\nabla\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{(1+\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))} * (\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) * (-y_i \boldsymbol{x}_i)$
This can then be equated to 0:
$\nabla\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{(1+\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))} * (\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) * (-y_i \boldsymbol{x}_i) = 0$
This can be considered as the updating step. If w' is the new weight, it can be written as follows:
$w' = w - \eta\nabla\widehat{\mathcal{R}}_{\log}(\boldsymbol{w})$
This is repeated for many iterations until training is completed.

(b) The code for the same has been attached. Similar to linear gradient descent function, this takes X, Y, learning rate and number of iterations as the input. The X matrix is pre-pended with a column of ones using torch.ones and torch.cat routines. The forward function provides the X*w value (which is predicted y). The loss function then returns the empirical risk as given in the question. torch.log and torch.exp are used to perform this function. The weight w is described as a tensor of zeroes and finally, the training takes place. After running num_iter iterations, the updated w is returned as the final answer.

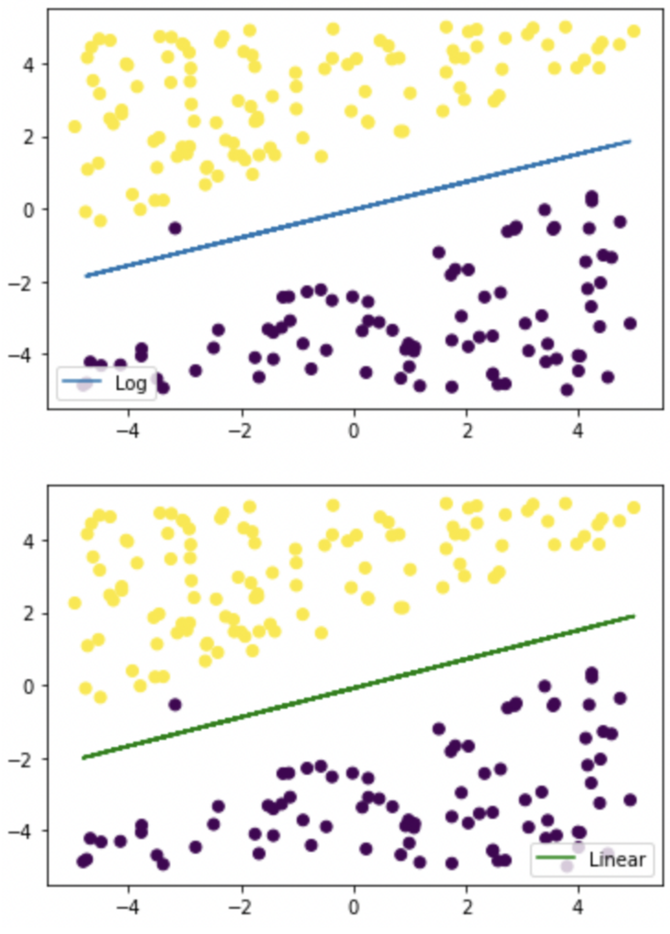


Figure 4: Logistic Regression vs Linear Regression

(c) The logistic_vs_ols() function displays the points as shown below: With 1000 iterations, the possibility of differentiating is difficult.

Note: The youtube source https://www.youtube.com/watch?v=EMXfZB8FVUAlist=PLqnslRFeH2U rcDBWF5mfPGpqQDSta6VK4 was used as reference to answer this homework.
The code with plotting functions failed to fetch full marks on Autograder. The initial code with required functions is run on it and the rest of the code, including plotting has been attached below.