CS 446 / ECE 449 — Homework 1

your NetID here

Version 1.3

Instructions.

- Homework is due **Tuesday, February 16, 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).
- When submitting to `hw1code`, only upload `hw1.py` and `hw1_utils.py`. Additional files will be ignored.

Version History.

1. Initial version.

1.1 Fixed typo in hint for 1c.

1.2 Clarified wording of 1a, 1b.

1.3 Fixed bolding of vectors in 2c.

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 $\hat{\boldsymbol{w}}_{\text{ols}} = \boldsymbol{X}^+ y$.

Remark: It's also not hard to show that if 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) We have the squared loss as

$$\begin{aligned} \frac{1}{2n} \sum_{i=1}^{n} (x^\top w - y_i)^2 &= \frac{1}{2n} \sum_{i=1}^{d} \sum_{j=1}^{n_i} (e_i^\top w - y_{i_j})^2 \\ &= \frac{1}{2n} \sum_{i=1}^{d} \sum_{j=1}^{n_i} (w_i - y_{i_j})^2 \end{aligned}$$

Taking the partial derivative and setting it equal to zero

$$\begin{aligned}
\frac{\partial MSE}{\partial w_i} &= \frac{1}{n}\sum_{j=1}^{n_i}(w_i - y_{i_j}) = \frac{1}{n}\sum_{j=1}^{n_i} w_i - \frac{1}{n}\sum_{j=1}^{n} y_{i_j} \\
&\Longrightarrow \sum_{j=1}^{n_i} w_i = \sum_{j=1}^{n_i} y_{i_j} \\
&\Longleftrightarrow w_i = \frac{1}{n_i}\sum_{j=1}^{n_i} y_{i_j}
\end{aligned}$$

We know this is the minimum as the squared loss is convex with respect to the parameters.

(b) There exists a w such that $\|Xw - y\|^2 = 0$ if and only if y is in the column space of X. But the set $\{u_i\}_{i=1}^r$ forms a basis for the column space of X (write X in the full SVD $X = U\Sigma V^\top$; then V is invertible and $U\Sigma$ is a matrix with the first r columns of U multiplied by the nonzero singular values). Thus, there exists a w such that $\|Xw - y\|^2 = 0$ if and only if y is a linear combination of the $\{u_i\}_{i=1}^r$.

To show the problem statement algebraically, let $y = \sum_{i=1}^r a_i u_i$ for some constants a_i possibly zero. Then

$$\begin{aligned}
\hat{w}_{\text{ols}} = X^+ y &= \left(\sum_{i=1}^r v_i u_i^\top / s_i\right)\left(\sum_{j=1}^r a_i u_i\right) \\
&= \sum_{i=1}^r \frac{a_i}{s_i} v_i. && \text{(orthonormality of the } u_i\text{'s)}
\end{aligned}$$

We then have

$$\begin{aligned}
\|X\hat{w}_{\text{ols}} - y\|^2 &= \left\| \left(\sum_{i=1}^r s_i u_i v_i^\top\right)\left(\sum_{j=1}^r \frac{a_i}{s_i} v_i\right) - \sum_{i=1}^r a_i u_i \right\|^2 \\
&= \left\| \sum_{i=1}^r a_i u_i - \sum_{i=1}^r a_i u_i \right\|^2 && \text{(orthonormality of the } v_i\text{'s)} \\
&= 0.
\end{aligned}$$

To show the other direction (though this is not necessary for the problem statement): if y isn't a linear combination of the first r u_i's then it is a linear combination of all n u_i's in the full SVD with not all zero coefficients for the last $(n-r)$ u_i's. Plugging this linear combination of y in as before, the left-hand sum remains the same whereas the right hand sum has additional nonzero terms for the u_i with $i \in \{n-r, \ldots, n\}$.

(c) The $\{x_i\}$ spanning $\mathbb{R}^d$ implies $\text{rank}(X) = d$. Hence, X has d nonzero singular values and $X^\top X$ has d nonzero eigenvalues. As $X^\top X$ is a $d \times d$ symmetric matrix, it has exactly d real eigenvalues, so $\det(X^\top X) = s_1^2 \cdot \ldots \cdot s_d^2 \neq 0$ and $X^\top X$ is invertible.

To show the other direction, assume $X^\top X$ is invertible. As it is a $d\times d$ matrix, $\text{rank}(X^\top X) = d$. For any two matrices A, B we have $\text{rank}(AB) \leq \min\left\{\text{rank}(A), \text{rank}(B)\right\}$. So

$$d = \text{rank}(X^\top X) \leq \min\left\{\text{rank}(X^\top), \text{rank}(X)\right\} = \text{rank}(X)$$

But X is an $n \times d$ matrix, so $\text{rank}(X) \leq d$. So $d \leq \text{rank}(X) \leq d$ implies $\text{rank}(X) = d$ and the $\{x_i\}$ span $\mathbb{R}^d$.

(d) Such a matrix X must have rows spanning $\mathbb{R}^d$ by problem (c) (i.e. $X^\top$ must be onto). A square matrix is singular iff it has a nontrivial kernel, and $\ker(XX^\top) = \ker(X^\top)$, so we need a matrix X such that $X^\top$ is onto and $\ker(X^\top) \neq \{0\}$. Namely, we need a $d \times n$ matrix $X^\top$ with d linearly independent columns and $n > d$, or equivalently, a matrix X with d independent rows and $n > d$. This completely characterizes the matrices X for which the conditions of (d) are satisfied. As an example, $X = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \\ a & b \end{smallmatrix}\right)$ satisfies the conditions for any a, b.

If we don't know that $\ker(X^\top) = \ker(XX^\top)$, we can again let $X = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \\ a & b \end{smallmatrix}\right)$ and partition the matrix with $R = \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$.

$$X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ a & b \end{pmatrix} = \begin{pmatrix} I \\ R \end{pmatrix}$$

$$\implies XX^\top = \begin{pmatrix} I \\ R \end{pmatrix} \begin{pmatrix} I & R^\top \end{pmatrix} = \begin{pmatrix} I & R^\top \\ R & RR^\top \end{pmatrix}$$

Recalling that the determinant of a matrix is the product of its diagonal for an upper triangular matrix, setting $R = (0, 0)$ gives $XX^\top$ singular.

2. Loss functions/MLE.

Recall that maximum likelihood estimation is a technique used to maximize the probability of observed data given parameters w of a probability model: $P(\text{data}|w)$. In machine learning, we want to find the most likely set of parameters w given the observed data. This can be expressed in terms of the likelihood function $\mathcal{L}(w) = P(\text{data}|w)$ as follows:

$$\arg\max_w P(w|\text{data}) = \arg\max_w \frac{P(\text{data}|w)P(w)}{P(\text{data})} = \arg\max_w P(\text{data}|w)P(w). \tag{1}$$

Assuming all parameters $\boldsymbol{w}$ are equally likely, (1) reduces to $\arg\max_w P(\text{data}|\boldsymbol{w})$ (we operate under this assumption for parts (a) and (b) of this problem). We represent the probability our model with parameters $\boldsymbol{w}$ assigns to data $(\boldsymbol{x}_i, y_i)_{i=1}^n$ by the likelihood function $\mathcal{L}(w) = \prod_{i=1}^n p_w(y_i|\boldsymbol{x}_i)$ where $p_w(y_i|\boldsymbol{x}_i)$ is our model's estimated probability of label y_i given $\boldsymbol{x}_i$.

(a) Suppose we are implementing a multiclass classifier with k target classes. Call the target class of the i'th training example $t_i \in \{1, \ldots, k\}$. We can encode the label $\boldsymbol{y}_i$ of the i'th training example as a one-hot vector with a 1 at index t_i and 0s elsewhere. Consider the likelihood function

$$\mathcal{L}(w) = \prod_{i=1}^n \prod_{j=1}^k p_w(t_j|\boldsymbol{x}_i)^{y_{i_j}},$$

where y_{i_j} is the j'th element of the vector $\boldsymbol{y}_i$. Show that maximizing this likelihood function is equivalent to minimizing the cross entropy loss (from Lecture 4) over the dataset

$$-\sum_{i=1}^n \sum_{j=1}^k y_{i_j} \log(p_w(t_j|\boldsymbol{x}_i)).$$

Remark: Why the double product? The inner one, $\prod_{j=1}^k p_w(t_j|\boldsymbol{x}_i)^{y_{i_j}}$, is just a clever way of writing the probability of the i'th training example: all of the exponents y_{i_j} will be zero except for y_{t_i}, which is 1. Therefore, the product reduces to the probability of the true label given the input.

(b) Recall that for logistic regression we minimize the logistic loss $\ell_{\text{logistic}}(y\boldsymbol{w}^\top \boldsymbol{x})$ when we model the probability of a positive label by $p_w(1|\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^\top \boldsymbol{x}}}$ and the labels $y \in \{-1, 1\}$. Another common loss function for logistic regression is binary cross entropy, defined by

$$-y\log(p_w(1|\boldsymbol{x})) - (1-y)\log(1 - p_w(1|\boldsymbol{x})),$$

where the labels $y \in \{0, 1\}$ (the cross entropy derived in part (a) is a generalization of this). Show that $\ell_{\text{logistic}}(y\boldsymbol{w}^\top \boldsymbol{x})$ with $y = -1$ is equal to binary cross entropy with $y = 0$ and that $\ell_{\text{logistic}}(y\boldsymbol{w}^\top \boldsymbol{x})$ with $y = 1$ is equal to binary cross entropy with $y = 1$.

Remark: This shows that the loss functions are equivalent up to a relabeling.

(c) Now we will drop the assumption that all parameters $\boldsymbol{w}$ are equally likely. Specifically, assume that the w_i are i.i.d. and follow a normal (Gaussian) distribution with mean 0 and variance $1/\lambda$: $w_i \sim \mathcal{N}(0, 1/\lambda)$. Independence of the w_i implies $P(\boldsymbol{w}) = \prod_{i=1}^d P(w_i)$. Further, assume that $y_i|\boldsymbol{x}_i, \boldsymbol{w}$ follows a normal distribution with mean $\boldsymbol{w}^\top \boldsymbol{x}_i$ and variance 1: $y_i|\boldsymbol{x}_i, \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}_i, 1)$. Show that

$$\arg\max_{\boldsymbol{w}} \prod_{i=1}^n p_{\boldsymbol{w}}(y_i|\boldsymbol{x}_i)P(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left\{ \frac{1}{2}\sum_{i=1}^n (\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)^2 + \frac{\lambda}{2}\sum_{j=1}^d w_j^2 \right\}.$$

Remark: This is an interpretation of the regularization term for ridge regression. The probability of very positive or very negative w_i is low if w_i follows a normal distribution, which is reflected by a larger penalty $\sum_i w_i^2$. (Note that we also dropped the $1/n$ factor of the empirical risk by maximum likelihood convention).

Solution.

(a) Our model's probability of the training data is $L(w) = \prod_{i=1}^n p_w(t_i|x_i)$. This is equal to the double product expression as y_i is encoded as a one-hot vector with $y_{i t_i} = 1$ (all non-true-label probabilities have a zero in the exponent).

Now

$$\begin{aligned}
\arg\max_w \prod_{i=1}^n \prod_{j=1}^k p_w(t_j|x_i)^{y_{i_j}} &= \arg\max_w \log\left(\prod_{i=1}^n \prod_{j=1}^k p_w(t_j|x_i)^{y_{i_j}}\right) \\
&\qquad\qquad\text{(log is monotonically increasing)} \\
&= \arg\max_w \sum_{i=1}^n \sum_{j=1}^k y_{i_j} \log(p_w(t_j|x_i)) \qquad \text{(log properties)} \\
&= \arg\min_w - \sum_{i=1}^n \sum_{j=1}^k y_{i_j} \log(p_w(t_j|x_i)) \\
&\qquad\qquad (\arg\max_w f(w) = \arg\min_w -f(x))
\end{aligned}$$

(b) We start with the logistic loss.

$$\begin{aligned}
\ell_{\text{logistic}}(yw^\top x) &= \log(1 + e^{-yw^\top x}) \\
&= \begin{cases} \log(1 + e^{-w^\top x}) & \text{positive class} \\ \log(1 + e^{w^\top x}) & \text{negative class} \end{cases}
\end{aligned}$$

Now for binary cross entropy. With $y = 1$ for the positive class

$$\begin{aligned}
&- y\log(p_w(1|x)) - (1-y)\log(1 - p_w(1|x)) \\
&\quad = -\log(p_w(1|x)) \\
&\quad = -\log\left(\frac{1}{1 + e^{-w^\top x}}\right) \\
&\quad = -\log(1) + \log(1 + e^{-w^\top x}) \\
&\quad = \log(1 + e^{-w^\top x})
\end{aligned}$$

With $y = 0$ for the negative class

$$\begin{aligned}
&- y\log(p_w(1|x)) - (1-y)\log(1 - p_w(1|x)) \\
&\quad = -\log(1 - p_w(1|x)) \\
&\quad = -\log\left(\frac{e^{-w^\top x}}{1 + e^{-w^\top x}}\right) \\
&\quad = -\log\left(\frac{1}{1 + e^{w^\top x}}\right) \qquad \text{(multiply num and denom by } e^{w^\top x}) \\
&\quad = -\log(1) + \log(1 + e^{w^\top x}) \\
&\quad = \log(1 + e^{w^\top x})
\end{aligned}$$

(c)

$$\begin{aligned}
\arg\max_w \prod_{i=1}^n p_w(y_i|x_i)P(w) &= \arg\max_w \prod_{i=1}^n p_w(y_i|x_i)\prod_{j=1}^d P(w_j) \\
&= \arg\max_w \left\{ \log\left(\prod_{i=1}^n p_w(y_i|x_i)\right) + \log\left(\prod_{j=1}^d P(w_j)\right)\right\} && \text{(monotonicity of log)} \\
&= \arg\max_w \left\{ \sum_{i=1}^n \log\left(p_w(y_i|x_i)\right) + \sum_{j=1}^d \log\left(P(w_j)\right)\right\} \\
&= \arg\max_w \left\{ \sum_{i=1}^n \log\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{(w^\top x_i - y_i)^2}{2}}\right) + \sum_{j=1}^d \log\left(\frac{1}{\sqrt{2\pi/\lambda}} e^{-\frac{w_j^2}{2/\lambda}}\right)\right\} \\
& && \text{(pdf of } \mathcal{N}(0,1/\lambda)) \\
&= \arg\max_w \left\{ \sum_{i=1}^n \left(\log\left(1/\sqrt{2\pi}\right) - \frac{(w^\top x_i - y_i)^2}{2}\right) + \sum_{j=1}^d \left(\log(1/\sqrt{2\pi/\lambda}) - \frac{\lambda w_j^2}{2}\right)\right\} \\
&= \arg\max_w \left\{ -\frac{1}{2}\sum_{i=1}^n (w^\top x_i - y_i)^2 - \frac{\lambda}{2}\sum_{j=1}^d w_j^2 \right\} && \text{(constants don't affect } \arg\max) \\
&= \arg\min_w \left\{ \frac{1}{2}\sum_{i=1}^n (w^\top x_i - y_i)^2 + \frac{\lambda}{2}\sum_{j=1}^d w_j^2 \right\}
\end{aligned}$$

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.

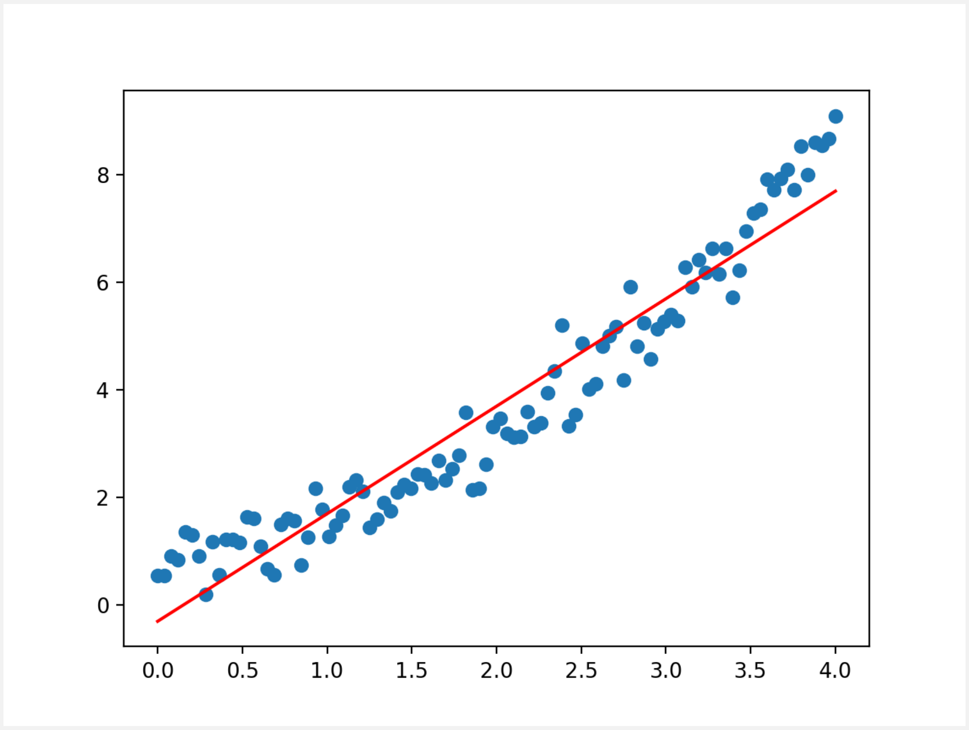
(a)
```
def linear_gd(X, Y, lrate=0.1, num_iter=1000):
    n = X.shape[0]
    X = torch.cat([torch.ones(n, 1), X], dim=1)
    w = torch.zeros(X.shape[1], 1)
    Y = Y.reshape([-1, 1])

    for t in range(num_iter):
        w = w - (1 / n) * lrate * (X.t() @ (X @ w - Y))

    return w
```

(b)
```
def linear_normal(X, Y):
    X = torch.cat([torch.ones(X.shape[0], 1), X], dim=1)

    return torch.pinverse(X) @ Y
```

(c) We will use the following Python snippet to generate the plot.

```
def plot_linear():
    X, Y = utils.load_reg_data()
    w = linear_normal(X, Y)

    XX = torch.cat([torch.ones(X.shape[0], 1), X], dim=1)
    fX = XX @ w
    plt.plot(X, fX, color='red')
    plt.scatter(X, Y)
    plt.show()

    return plt.gcf()
```

The resulting plot will be:



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

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

defined by

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

(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 using a nested for loop.

(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 best? 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 labels for the XOR points. Include a plot of contour lines that show how each model classifies points in your written submission. You may use `contour_plot()` in `hw1_utils.py`. Return the labels for both the linear model and the polynomial model in that order. Do both models correctly classify all points? (Note that red corresponds to larger values and blue to smaller values when using `contour_plot`).

Solution.

For the coding parts define a helper function `poly(X)`:

```
def poly(X): # Make quadratic features
    XX = torch.einsum('ij,ik->ijk', X, X) # Pairwise products per row
    indices = torch.triu_indices(X.shape[1], X.shape[1]) # No repetitions
    XX = XX[:, indices[0], indices[1]] # Matrix of quadratic features

    return torch.cat([torch.ones(X.shape[0], 1), X, XX], dim=1)
```

(a) $\phi(x) = (1, x_1, x_2, x_3, x_1x_1, x_1x_2, x_1x_3, x_2x_2, x_2x_3, x_3x_3)$

(b)
```
def poly_gd(X, Y, lrate=0.01, num_iter=1000):
    n = X.shape[0]
    X = poly(X)
    w = torch.zeros(X.shape[1], 1)

    for t in range(num_iter):
        w = w - lrate / n * X.t() @ (X @ w - Y)

    return w
```

(c)
```
def poly_normal(X,Y):
    return torch.pinverse(poly(X)) @ Y
```

(d)
```
def plot_poly():
    X, Y = utils.load_reg_data()

    w = poly_normal(X, Y)
    fX = poly(X) @ w

    plt.plot(X, fX, color='red')
    plt.scatter(X, Y)
    plt.show()

    return plt.gcf()
```

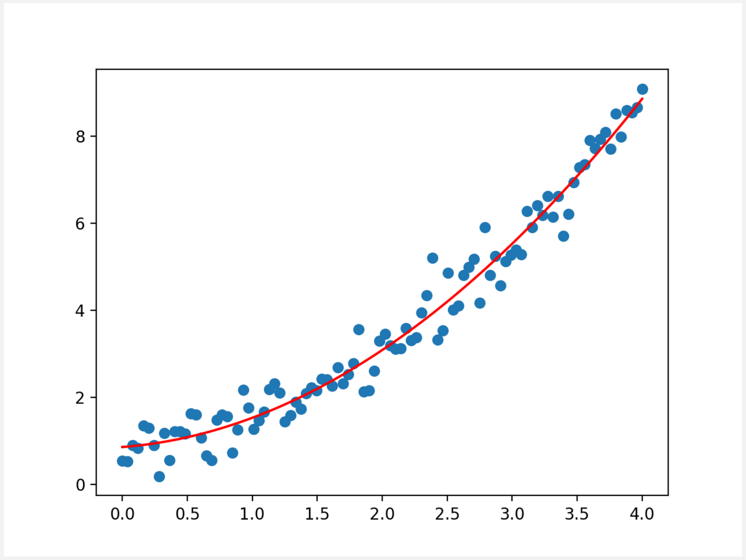
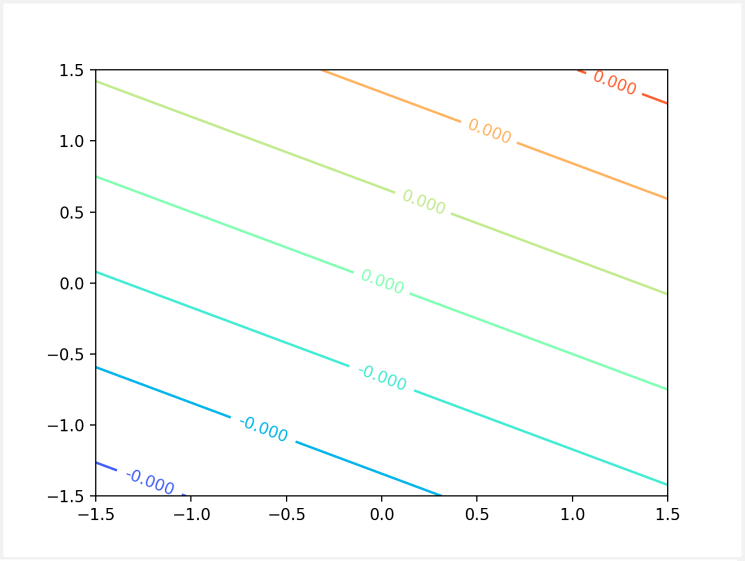


Figure 1: part (d) figure

The quadratic expanded model appears to approximate the data better because it yields a lower risk without overfitting the points.

(e)
```
def poly_xor():
    X, Y = utils.load_xor_data()

    w_linear = linear_normal(X, Y)
    w_poly = poly_normal(X, Y)

    def prependOnes(M):
        return torch.cat([torch.ones(M.shape[0], 1), M], dim=1)

    linear_pred = lambda features: prependOnes(features) @ w_linear
    poly_pred = lambda features: poly(features) @ w_poly

    utils.contour_plot(-1.5, 1.5, -1.5, 1.5, linear_pred)
    utils.contour_plot(-1.5, 1.5, -1.5, 1.5, poly_pred)

    return linear_pred(X), poly_pred(X)
```



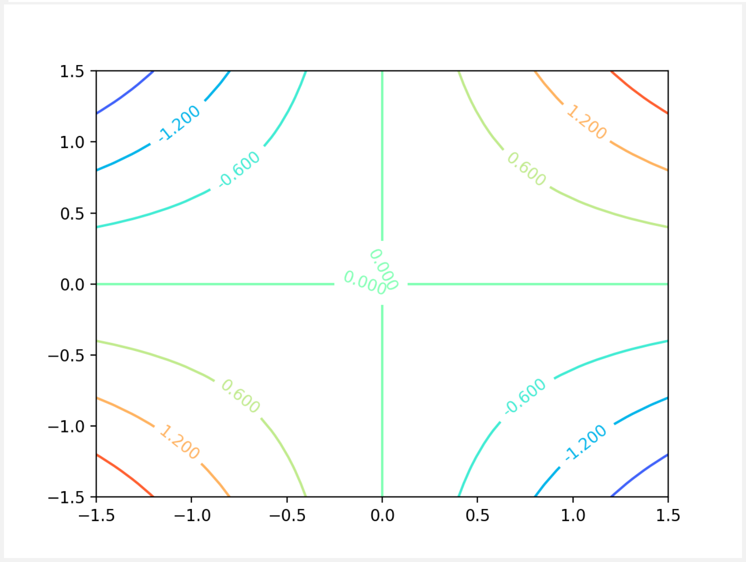


Figure 2: part (e) figure

Based on these plots we can see that the model with quadratic expanded features correctly classifies all points, but the linear model fails to correctly classify the points (predicting zero everywhere).

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.`

Solution.

(a) Let $\ell(x) = \ln(1+\exp(-x))$. First we compute the gradient:

$$\begin{aligned}
\nabla_w \widehat{\mathcal{R}}_{\log}(w) &= \nabla_w \frac{1}{n}\sum_{i=1}^{n} \ell(w^\top x_i y_i) \\
&= \frac{1}{n}\sum_{i=1}^{n} \nabla_w \ell(w^\top x_i y_i) \quad \boxed{\text{Derivative is linear}} \\
&= \frac{1}{n}\sum_{i=1}^{n} \ell'(w^\top x_i y_i) x_i y_i \quad \boxed{\text{Chain Rule}} \\
&= -\frac{1}{n}\sum_{i=1}^{n} \frac{\exp(-y_i w^\top x_i)}{1+\exp(-y_i w^\top x_i)} x_i y_i \quad \boxed{\text{Plug in derivative}}
\end{aligned}$$

The gradient update is then:

$$w' := w - \eta \nabla \widehat{\mathcal{R}}_{\log}(w)$$

Substituting our gradient in we obtain the update:

$$w' := w + \eta \frac{1}{n}\sum_{i=1}^{n} \frac{\exp(-y_i w^\top x_i)}{1+\exp(-y_i w^\top x_i)} x_i y_i$$

(b)
```
def logistic(X, Y, lrate=.01, num_iter=1000):
    X = torch.cat([torch.ones(X.shape[0], 1), X], dim=1)
```

```
        Z = -Y * X
        # deriv of logistic loss
        lp = lambda x: torch.exp(x) / (1 + torch.exp(x))
        g = lambda w: (lp(Z @ w).t() @ Z).t() / len(Z) # gradient of risk

        w = torch.zeros(X.shape[1], 1)
        for t in range(num_iter):
            w = w - lrate * g(w)

        return w
```

(c)
```
def logistic_vs_ols():
        X, Y = utils.load_logistic_data()

        w_linear = linear_normal(X, Y)
        w_logistic = logistic(X, Y)

        pos_indices = torch.tensor([i for i in range(len(Y)) if Y[i] > 0])
        neg_indices = torch.tensor([i for i in range(len(Y)) if Y[i] < 0])

        plt.scatter(X[pos_indices, 0], X[pos_indices, 1], color='red')
        plt.scatter(X[neg_indices, 0], X[neg_indices, 1], color='blue')

        x_points = torch.linspace(-5, 5, 2)

        plt.plot(x_points, -(w_linear[1] * x_points + w_linear[0]) / w_linear[2],
                 label='linear')
        plt.plot(x_points, -(w_logistic[1] * x_points +
    w_logistic[0]) / w_logistic[2],
                 label='logistic')
        plt.legend(loc='upper right')
        plt.show()

        return plt.gcf()
```

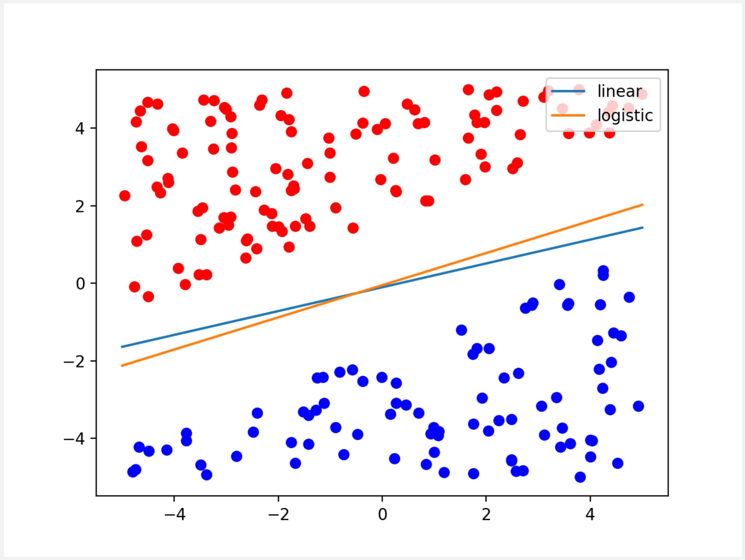


Figure 3: part (c) figure

As is shown in the figure, both models are able to classify all of the points in the training set correctly. However, the logistic regression model is still likely the better choice for this problem because it yields a much larger margin than the linear regression model. That is, in the logistic regression model the distance between the decision boundary and the support vectors (i.e. the closest points to the boundary) is maximized.