CS 446 MJT — Homework 1

your NetID here

Version 1

Instructions.

- Homework is due **Tuesday, February 5, at 11:59pm**; 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., rather than implementing an algorithm, you keep re-submitting the assignment to the auto-grader, eventually completing a binary search for the answers).
- In this assignment, for all code unless otherwise specified please return your output as a NumPy array.

1. **Decision Trees.**

Consider the training data given in Figure 1.

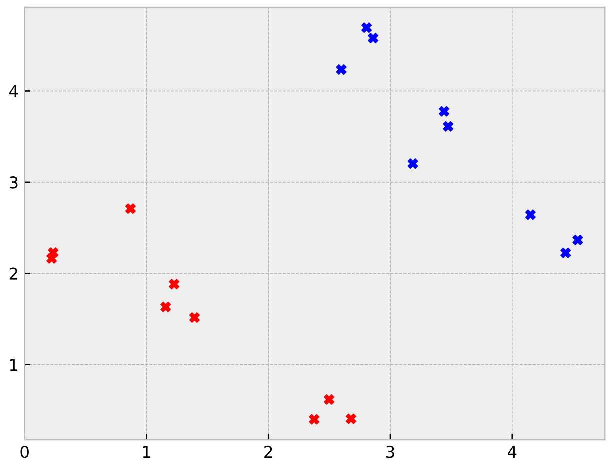

Figure 1: Training data

(a) Describe a decision tree of depth one with integral and axis-aligned decision boundaries for this data set, with training error at most $\frac{1}{6} = .166\ldots$.

(b) Describe a decision tree with integral and axis-aligned decision boundaries for this data set, with zero training error.

(c) Describe a decision tree with integral and axis-aligned decision boundaries for this data set, with zero training error such that the testing error rate is large when given the testing set of Figure 2.

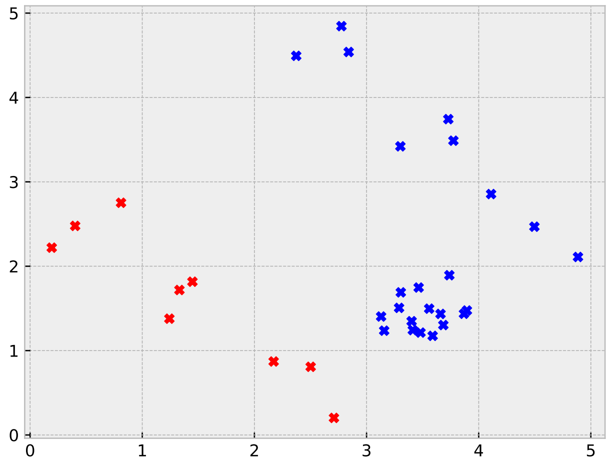

Figure 2: Testing data

Solution. *(Your solution here.)*

2. **Linear Regression.**

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

(a) Implement the linear regression method using gradient descent in `linear_gd(X, Y, lrate, num_iter)` function in `hw1.py`. You are given a training set `X` as input and training labels `Y` as input along with a learning rate `lrate` and maximum number of iterations `num_iter`. Using gradient descent find parameters w that minimize the empirical risk $\widehat{\mathcal{R}}(w)$. One iteration is equivalent to one full data gradient update step. Use a learning rate of `lrate` and only run for `num_iter` iterations. Use $w = 0$ as your initial parameters, and return your parameters w as output. (Note: gradient descent will be covered in lecture 5.)

(b) Implement linear regression by setting the gradient to zero and solving for the variables, in `linear_normal(X,Y)` function in `hw1.py`. You are given a training set `X` as input and training labels `Y` as input. (Lectures 3-4 give a few ways to get an answer here.) Return your parameters w as output.

(c) Implement the `plot_linear()` function in `hw1.py`. Use the provided function `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.

Solution. *(Your solution here.)*

3. **Singular Value Decomposition.**

Recall, as detailed in lecture 3, that for every matrix $A \in \mathbb{R}^{n\times d}$, there are matrices $U \in \mathbb{R}^{n\times n}$, $S \in \mathbb{R}^{n\times d}$ and $V \in \mathbb{R}^{d\times d}$ such that $A = USV^\top$ and S is a diagonal matrix and U and V are orthonormal matrices, that is $U^{-1} = U^\top$ and $V^{-1} = V^\top$ (i.e. its inverse is equal to its transpose). (A convenient alternative notation, as discussed in lecture, is $A = \sum_{i=1}^r s_i u_i v_i^\top$.)

(a) Let $A \in \mathbb{R}^{n\times n}$ be a square matrix and consider its singular value decomposition $USV^\top$ with values $s_1, \dots, s_n$ on the diagonal of S. Show that A is invertible if and only if $s_i \neq 0$ for all $i \in \{1, \dots, n\}$.

(b) Show that for all $A \in \mathbb{R}^{n\times d}$ and all positive $\lambda \in \mathbb{R}_{>0}$, the matrix $(A^\top A) + \lambda I$ is invertible.

(c) Prove that $\lim_{\lambda\downarrow 0}(A^\top A + \lambda I)^{-1}A^\top \to A^+$, where A^+ is the pseudoinverse of the matrix A. That is to show that every entry of the matrix $(A^\top A + \lambda I)^{-1}A^\top$ converges to the corresponding entry in A^+ as λ vanishes.

Solution. *(Your solution here.)*

4. **Polynomial Regression.**

In problem 2 you constructed a linear model $w^\top x = \sum_{i=1}^d x_i w_i$. 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_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:

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

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

(b) Implement the `poly_gd()` function in `hw1.py`. The input is in the same format as it was in problem 2. Use this training set to implement gradient descent to determine the parameters w. Use $w = 0$ as your initial parameters. Return your w parameters as output. Please return your parameters in the exact order mentioned here (bias, linear, and then quadratic). For example, if $d = 3$ then you would return $(w_0, w_{01}, w_{02}, w_{03}, w_{11}, w_{12}, w_{13}, w_{22}, w_{23}, w_{33})$.

(c) Implement the `poly_normal` function in `hw1.py`. You are given the same data set as from part (b), but this time you will determine the w parameters by solving the normal equations. Return your w parameters as output. Again, return these parameters in the same order you returned them for part (b).

(d) Implement the `plot_poly()` function in `hw1.py`. Use the provided function `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 2. Which model appears to approximate the data best? Provide a justification for your answer.

(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 `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` to help you plot the contour lines. As output, return both the labels for the linear model and the polynomial model in that order. Do both models correctly classify all points?

Solution. *(Your solution here.)*

5. **Nearest Neighbor.**

   (a) Implement the 1-nearest neighbor algorithm in the `nn()` function in `hw1.py`. In the starter code you are given three NumPy arrays as input:

   - `X` - training set
   - `Y` - training labels
   - `X_test` - testing set

   Use the training set to determine labels for the testing set. Return the labels for the testing set as determined by your nearest neighbor implementation.

   (b) Plot the Voronoi diagram of your nearest neighbor results. Use the data set returned from `utils.load_nn_data()` to make this plot. You may use the function `utils.voronoi_plot()` provided to you in `hw1_utils.py` to help generate the diagram. There is no need to submit code for this part, only submit the plots in the written portion.

   (c) Implement the `nn_iris()` function in `hw1.py`. Here you will use your nearest neighbor implementation on the Iris data set provided by the scikit-learn library (which can be installed via "`pip3 install scikit-learn`"). Use the `utils.load_iris_data()` function to load the data set, and then split the data set into a testing set and training set. Take the first 30% of the data set to be the testing set, and the rest of the data set to be the training set. Run your `nn()` function on this data set and return your classification accuracy as output. Report your classification accuracy in your written submission.

Solution. *(Your solution here.)*

6. **Logistic Regression.**

Recall the empirical risk $\widehat{\mathcal{R}}$ for logistic regression (as presented in lectures 5-6):

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

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 w, new parameters w', number of examples n, and training examples x_i. Show all of your steps.

(b) Implement the `logistic()` function in `hw1.py`. You are given a training set `X` as input and training labels `Y` as input along with a learning rate `lrate` and maximum number of iterations `num_iter`. Implement gradient descent in order to find parameters w that minimize the empirical risk $\widehat{\mathcal{R}}_{\log}(w)$. One iteration is equivalent to one full data gradient update step. Return your parameters w as output. You may use PyTorch to handle the gradient computation for you. Use $w = 0$ as your initial parameters. Use a learning rate of `lrate` and only run for `num_iter` iterations.

(c) Now implement the `logistic_vs_ols()` function in `hw1.py`. This time you are only given the training set `X` and training labels `Y` as input. Run `logistic(X,Y)` from part (b) taking `X` and `Y` as input to obtain parameters w (use the defaults for `num_epochs` and `lrate`). Also run `linear_gd(X,Y)` from problem 2 also taking `X` and `Y` as input to obtain parameters w. Plot both lines generated by logistic regression and least squares along with the data `X`. Which model appears to classify the data better? Provide an explanation for why you believe your choice is the better classifier for this problem.

Solution. *(Your solution here.)*