EMORY UNIVERSITY
Department of Computer Science
CS 334 Section 2 — Machine Learning
Fall 2024

Homework 2, Issued: Fri. 09/13, Due: Fri. 09/27 at 11:59pm

Submission Instructions: The homework is due on Gradescope in two parts.

- **Upload PDF to HW2-Written**: Create a single high-quality PDF with your answers to the non-coding problems. Your submission may be typed or handwritten (can be scanned or from a note-taking software), but the pages must be tagged with the relevant questions appropriately on Gradescope. You do not need to include code in the PDF unless otherwise instructed.
- **Submit code to the HW2-Code**: Questions marked with 🛠 on the left margin are graded by a Python autograder. Your submission must include only the following files: `perceptron.py`, `linear_regression.py`, `README.txt` (no data files please). You may submit multiple times but be sure to upload *ALL* files when you re-submit; only the latest submission will be considered. The `README.txt` file must contain *a **SIGNED** honor code statement* that reads as follows:

```
THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING CODE
WRITTEN BY OTHER STUDENTS OR LARGE LANGUAGE MODELS SUCH AS CHATGPT.
/* Your_Name_Here */
I collaborated with the following classmates for this homework:
<names of classmates>
```

Quick note about code runtimes: a correct implementation should take <1 minute to run all experiments in this homework.

1 Decision Boundaries (12 pts)

Note: In this course, we use the convention that points on the decision boundary are misclassified.

(a) Consider the AND function defined over three binary variables: $f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3)$. We aim to find a $\vec{\theta} \in \mathbb{R}^3$ such that for any $\vec{x} = [x_1, x_2, x_3]^T$, where each $x_d \in \{0, 1\}$:

$$\vec{\theta} \cdot \bar{x} + b > 0 \text{ when } f(x_1, x_2, x_3) = 1, \text{ and}$$
$$\vec{\theta} \cdot \bar{x} + b < 0 \text{ when } f(x_1, x_2, x_3) = 0.$$

i. (2pts) If $b = 0$ (i.e., no offset), would it be possible to learn such a $\bar{\theta}$? Explain.

ii. (2pts) How about if a non-zero offset is allowed? Provide an example of such $\bar{\theta}$ and b, or explain why it's not possible.

(b) You are given the following labeled data points:

- Positive examples: $[1, 1]$ and $[-1, -1]$.
- Negative examples: $[-1, 0]$ and $[0, 1]$.

For each of the following parameterized families of classifiers, give an example (find the parameters) of a family member that can correctly classify the above data, or explain why no such family member exists. Make sure to specify the direction of the negative & positive decision.

i. (2pts) Inside or outside of a circle centered at the origin with radius r.

ii. (2pts) Above or below a line through the origin with normal $[a, b]$.

iii. (2pts) Inside or outside of a square centered at $[a, b]$ with sides parallel to the axes and side length s

iv. (2pts) Inside or outside of a square centered at the origin with side length s and counter-clockwise rotation α, where $\alpha = 0$ implies sides parallel to the axes

2 Perceptron Algorithm with Offset (20 pts)

Consider a ***sequence*** of 2-dimensional data points, $\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(n)}$ and their corresponding labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$. Recall the perceptron algorithm updates the parameters whenever $y^{(i)} \neq h(\vec{x}^{(i)}; \vec{\theta})$ where $h(\vec{x}^{(i)}; \vec{\theta}) = \text{sign}(\vec{\theta} \cdot \vec{x}^{(i)} + b)$. Assume that the points are linearly separable, and that both $\vec{\theta}$ and b are initialized to zero. Let α_i denote the number of times $\vec{x}^{(i)}$ is misclassified during training.

(a) (2pts) Express the parameters $\vec{\theta}$ and b of the final decision boundary for the perceptron algorithm in terms of α_i, $\vec{x}^{(i)}$ and $y^{(i)}$.

(b) (2pts) Show that the shortest **signed** distance from the boundary to the origin is equal to $\frac{b}{||\vec{\theta}||}$.

🛠 (c) (7pts) We have provided you with skeleton code in `perceptron.py`. Implement the helper function `all_correct(X,y,theta,b)` and the algorithm function `perceptron(X,y)`, following the function specifications in the skeleton code. Do not shuffle the points.

(d) (5pts) We've also given you a dataset `classification.csv`. Report the $\vec{\theta}$ and b produced by your implementation and $\vec{\alpha}$, the number of times each point is misclassified. Your answers for $\vec{\theta}$, b, and $\vec{\alpha}$ should match with the expressions you derived in (a).

(e) (4pts) Given a set of linearly separable points, does the order in which the points are presented to the algorithms affect whether or not the algorithm will converge? In general, could the order affect the total number of mistakes made?

3 Linear Regression (30 + 20 + 18 pts)

You are provided with skeleton code `linear_regression.py` and data files, `linreg_train.csv` and `linreg_validation.csv`, which specify a linear regression problem for a polynomial. In each csv file, the first column specifies the output ($y^{(i)} \in \mathbb{R}$), and the second column specifies the input ($x^{(i)} \in \mathbb{R}$). There is one training/validation example per row.

You may find it useful to generate 2D-plots for the training data and the output of regression functions.

Recall that for linear regression, the empirical risk with **squared loss** is:

$$R_N(\vec{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \frac{(y^{(i)} - \vec{\theta} \cdot \vec{x}^{(i)})^2}{2}$$

3.1 Linear Regression - Optimization Method (30 pts)

(a) (4pts) Implement the function `generate_polynomial_features(X,M)` according to the specification in the skeleton code. This function transforms each example $x^{(i)}$ into an $M+1$ dimensional feature vector, $\phi(x^{(i)})$. In part 3.1 you will explore a solution based on a first degree polynomial. However, the function should be general enough to handle any $M \geq 0$ for latter parts.

(b) (0pt) Implement the helper function `calculate_squared_loss` according to the specification, which computes the empirical risk with squared loss defined above. Note that this question is not graded for credit. You will call this helper function in subsequent parts, so we've created public test cases to help you debug.

(c) You will now implement three different optimization methods to find the coefficients θ_1 and θ_0 (slope and intercept, respectively) that minimize the **squared loss** for a first degree polynomial $\hat{y} = \theta_1 x + \theta_0$.

- (3pts) Implement `ls_gradient_descent(X,y)`.
- (3pts) Implement `ls_stochastic_gradient_descent(X,y)`.
- (3pts) Implement `ls_closed_form_solution(X,y)` (ignore `reg_param` for now).

Important note:

- For the family of gradient descent algorithms, check for convergence after each epoch (one pass through the entire training set).
- For SGD, do NOT shuffle the points after each epoch.
- The `prev_loss` and `new_loss` in the skeleton code refer to empirical risk for linear regression as defined above.
- Use the convergence criteria specified in the code: the algorithm should terminate when there is either marginal improvement in the loss ($< 10^{-10}$) during a single iteration, or after $1,000,000$ iterations — whichever happens first.
- For the closed form solution, if you encounter numerical stability issues in the calculation of matrix inverse, consider using the pseudoinverse operation `np.linalg.pinv()`.

(d) (4pts) Use the functions you implemented above to find the coefficients θ_1 and θ_0 that minimize the squared loss for a first degree polynomial ($M = 1$). For GD and SGD, you need to specify a learning rate (or step size) η. Try different values of $\eta \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Report your results in the following table. Here "# iterations" refers to the number of $\bar{\theta}$ updates.

Algorithm	η	θ_0	θ_1	# iterations	Runtime (s)
GD	10^{-4}				
GD	10^{-3}				
GD	10^{-2}				
GD	10^{-1}				
SGD	10^{-4}				
SGD	10^{-3}				
SGD	10^{-2}				
SGD	10^{-1}				
Closed form	-			-	

(e) (3pts) Compare GD vs SGD: specifically, comment on runtime, number of iterations, and resulting coefficients at convergence.

(f) (2pts) In your experiments, how does the runtime of the closed-form solution compare to SGD? Which learning rate used in SGD produces the coefficients closest to the closed form solution?

(g) (3pts code + 5pts written) Propose a learning rate η_k that is a function of k (the number of iterations) and implement it in your code for SGD if the input argument `learning_rate='adaptive'`. How long does the algorithm (in terms of runtime and number of iterations) take to converge with your proposed learning rate? Are the coefficients produced by using your proposed learning rate close to the closed-form solution? How does the performance compare with SGD/GD with constant learning rates?

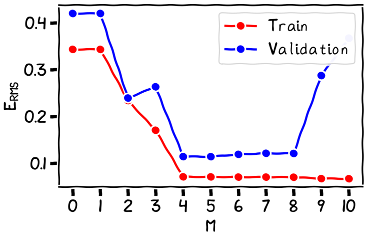
3.2 Linear Regression - Overfitting & Regularization (20 pts)

Next, you will investigate the problem of overfitting. Here, we observe overfitting as we increase the degree of the polynomial, M. We evaluate solutions using Root-Mean-Square (RMS) Error, defined as

$$E_{\text{RMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \vec{\theta}\cdot\phi(x^{(i)})\right)^2}$$

(a) (0pt) Implement the function `calculate_RMS_Error(X,y,theta)` according to the specification given in the skeleton code. Note that E_{RMS} is related to, but different from, empirical risk with squared loss.

(b) (5pts) Using `ls_closed_form_solution()`, find the coefficients that minimize the empirical risk with **squared loss** for an M^{th} degree polynomial (for $M = 0\ldots10$) for the training data. Now use `calculate_RMS_Error()` to calculate the RMS Error for each setting of M, on the training data and on the validation data (separately). Plot E_{RMS} against M for both training data and validation data (in the same graph) and include it in your write-up. To help you debug, your graph should look similar to the following:



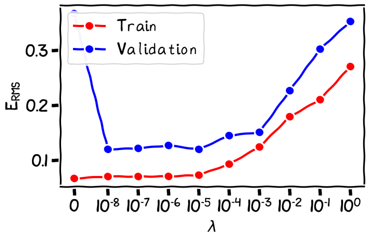
(c) (3pts) Which degree polynomial would you say best fits the data? Is there any evidence of underfitting / overfitting? Use your generated plot to justify your answer.

(d) (3pts) Modify your implementation of `ls_closed_form_solution(X,y,reg_param=0)` from part 3.1 to incorporate L2-regularization. Specifically, use the following regularized objective function:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{(\vec{\theta}\cdot\phi(x^{(i)}) - y^{(i)})^2}{2} + \frac{\lambda}{2}\|\vec{\theta}\|_2^2$$

for optimizing the parameters $\vec{\theta}$.

(e) (5pts) Use your function from part (d) to find the coefficients that minimize the objective function for a tenth degree polynomial ($M = 10$) given regularization parameter $\lambda \in \{0, 10^{-8}, 10^{-7}, \dots, 10^{-1}, 10^{0}\}$ for the training data specified in `linreg_train.csv`. Now use these coefficients to calculate the RMS Error on both the training data and validation data as a function of λ and plot E_{RMS} against $\lambda \in \{0, 10^{-8}, 10^{-7}, \dots, 10^{-1}, 10^{0}\}$. Your plot should look something like the following:



(f) (2pts) Which value of λ appears to work the best? Explain your answer.

(g) (2pts) Based on your results, what values of M and λ would you use to model this dataset? If you feel that you can't make a call yet, describe any additional experiments you might want to run.

3.3 Weighted Linear Regression (18 pts)

Finally, consider a linear regression problem in which we want to "weight" individual training examples differently. Specifically, suppose we want to minimize the cost function for weighted linear regression

$$J(\vec{\theta}) = \sum_{i=1}^{N} w^{(i)} \left(\vec{\theta} \cdot \vec{x}^{(i)} - y^{(i)}\right)^2 + \lambda \|\vec{\theta}\|_2^2$$

where $D_N = \{\vec{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ is the set of training examples, λ is the regularization parameter, and $w^{(i)} \geq 0$.

(a) (2pts) Describe a situation in which it would be useful to weight examples differently during training.

(b) (4pts) Show that $J(\vec{\theta})$ can also be written as

$$J(\vec{\theta}) = (X\vec{\theta} - \vec{y})^T W (X\vec{\theta} - \vec{y}) + \lambda \vec{\theta}^T \vec{\theta}$$

for some suitable diagonal matrix W, where $X = [\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(N)}]^T$ and $\vec{y} = [y^{(1)}, y^{(2)}, \dots, y^{(N)}]^T$. Clearly state what W is.

(c) (5pts) Find the closed-form solution for $\vec{\theta}$ by taking the gradient of $J(\vec{\theta})$ with respect to $\vec{\theta}$ and setting that to zero. Express your answer in terms of X, $\vec{y}$, and W as defined in (b). Show your work.

(d) (4pts) Implement `weighted_ls_closed_form_solution(X,y,weights,reg_param=0)` using what you derived in (c).

(e) (3pts) Use your function from part (d), follow what you did in part 3.2 to find which degree polynomial best fits the data (assume no regularization). Do you get the same conclusion as before? Why do you think that is the case?

REMEMBER: Submit your completed assignment by 11:59pm on Sept. 27th to Gradescope.