CS 229, Winter 2025
Problem Set #1

Due Wednesday, January 22 at 11:59 pm on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Ed forum, at `https://edstem.org/us/courses/70557/discussion`. (3) This quarter, Winter 2025, all homework assignments must be submitted individually. However, students may work in groups. If you do so, make sure both names are attached to the submission. If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on the course website before starting work. (4) For the coding problems, you may not use any libraries except those defined in the provided `environment.yml` file. In particular, ML-specific libraries such as scikit-learn are not permitted. (5) To account for late days, the due date is Wednesday, January 22 at 11:59 pm. If you submit after Wednesday, January 22 at 11:59 pm, you will begin consuming your late days. If you wish to submit on time, submit before Wednesday, January 22 at 11:59 pm.

All students must submit an electronic PDF version of the written questions. We highly recommend typesetting your solutions via LaTeX, and we will award one bonus point for typeset submissions. To edit on Overleaf, upload the `pset1.zip` folder as new project. In the project view, click on the "Menu" button located on the top left corner. In the "Settings" subsection, you will need to set the "Main Document" to be "tex/ps1.tex". Also, make sure your **cursor is selected on ps1.tex** before your press the **Recompile button**. Type your response in the `-sol.tex` file version for every problem. Inside `text/ps1.tex` file, you will need to change `\def\solutions{0}` to `\def\solutions{1}` and then re-compile. To export the PDF file, click the downward arrow next to the **Recompile button**.

All students must also submit a zip file of their source code to Gradescope, which should be created using the `make_zip.py` script. You should make sure to (1) restrict yourself to only using libraries included in the `environment.yml` file, and (2) make sure your code runs without errors. Your submission may be evaluated by the auto-grader using a private test set, or used for verifying the outputs reported in the writeup.

1. **[30 points] Convergence and Learning Rate for Gradient Descent**

When running an optimization algorithm, finding a good learning rate may require some tuning. If the learning rate is too high, the Gradient Descent (GD) algorithm might not converge. If the learning rate is too low, Gradient Descent may take too long to converge. In this question, we will investigate some theoretical guarantees for the convergence of Gradient Descent on convex objective functions, and their implications on choosing the learning rate.

Recap and notation. Suppose we have a convex objective function $J(\theta)$. At each iteration, Gradient Descent updates the iterate $\theta^{[t]}$ as follows:

$$\theta^{[t]} = \theta^{[t-1]} - \alpha \nabla J(\theta^{[t-1]})$$

where α is the learning rate and $\nabla J(\theta^{[t-1]})$ is the gradient of $J(\theta)$ evaluated at $\theta^{[t-1]}$.

(a) **[8 points] Quadratic Objective, Scalar Variable**

In this part, consider the simple objective function with one-dimensional variable θ:

$$J(\theta) = \frac{1}{2}\beta\theta^2$$

where $\theta \in \mathbb{R}$ is our parameter and $\beta > 0$ is a positive, constant scalar. We initialize Gradient Descent at some initialization $\theta^{[0]} \neq 0$ and run Gradient Descent with learning rate α.

i. **Derive** the range of α such that the iterate of Gradient Descent converges in the sense that there exists a scalar $\theta^\dagger$ such that $\lim_{t\to\infty} |\theta^{[t]} - \theta^\dagger| = 0$. Provide the value of $\theta^\dagger$ when Gradient Descent converges and show that it's equal to the global minimum $\theta^* = \arg\min_\theta J(\theta)$. The range of α can potentially depend on β.

ii. Given a desired accuracy $\epsilon > 0$, for all the α in the range that you derived, **derive** the minimum number of iterations T required to reach a point $\theta^{[T]}$ such that $|\theta^{[T]} - \theta^*| \leq \epsilon$. T can potentially depend on ϵ, $\theta^{[0]}$, α, and β. Investigate the behavior of T and whether T is increasing or decreasing for different values of α. Briefly discuss why choosing an inappropriate α can cause the algorithm to take longer to converge.

Hint: Express $\theta^{[t]}$ in terms of $\theta^{[0]}$, α, t, and β.

Hint: $\lim_{t\to\infty} c^t = 0, \quad \forall c \in \mathbb{R}\ s.t.\ |c| < 1$

(b) **[4 points] Quadratic Objective, d-dimensional Variable**

In this part of the question, consider the objective function with d-dimensional variable θ:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{d} \beta_i \theta_i^2$$

where θ_i's are our parameters and $\beta_i \in \mathbb{R}\ s.t.\ \beta_i > 0$ are positive, constant scalars. Assume that we start from some $\theta^{[0]}$ where $\theta_i^{[0]} \neq 0$ for all i and run Gradient Descent with learning rate α. Derive the range of learning rate α such that Gradient Descent converges in the sense that there exists a vector $\theta^\dagger \in \mathbb{R}^d$ such that $\lim_{t\to\infty} \|\theta^{[t]} - \theta^\dagger\|_2 = 0$. Provide the value of $\theta^\dagger$ when Gradient Descent converges. The range of α can potentially depend on β_i where $i \in \{1, \ldots, d\}$.

(c) **[4 points] Coding Question: Quadratic Multivariate Objective**

Let the objective function be

$$J(\theta) = \theta^\top A \theta$$

where $A \in \mathbb{R}^{2\times 2}$ is a 2×2 real, positive semi-definite matrix. Do the following exercises:

i. Implement the `update_theta` and `gradient_descend` function in `src/gd_convergence/experiment.py`. You can stop the Gradient Descent algorithm when either of the following condition is satisfied:

   A. $|J(\theta^{[t]}) - J(\theta^{[t-1]})| < \epsilon$ where $\epsilon = 10^{-50}$ is given as the function parameter. This is when we assume the algorithm converged.

   B. $J(\theta^{[t]}) > 10^{20}$. This is to prevent an infinite loop when the algorithm does not converge.

   To test your implementation, run `python src/gd_convergence/experiment.py`, which checks that your θ (approximately) converges to the optimal value.

   Note that we have provided you a matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ and $\theta^{[0]} = [-1, 0.5]$ at the beginning of the file for you to experiment with. Therefore, the objective function is a special case of the objective function in part (b) with dimension $d = 2$. Check if your theoretical derivation of the feasible range of the learning rate indeed matches empirical observations.

ii. Now, suppose we rotate the matrix A, what do we observe? Plot the trajectories of the Gradient Descent algorithm using the following learning rates: 0.05, 0.1, 0.2, 0.3, 0.4, 0.45, 0.5, and 1. We have provided the rotation and plotting function for you. You need to simply run `python src/gd_convergence/plotting.py lr1 lr2 lr3 ...` with `lr*` replaced with desired learning rates. Include the output file `trajectories.png` and `trajectories_rotated.png`, as well as a brief discussion on the printed output of `plotting.py` in the write-up.

Remark: Setting the learning rate too high will cause the objective function to not converge. The convergence properties of these learning rates are also rotational invariant. We will show this formally in the next part of the problem.

(d) **[12 points] Convergence of Gradient Descent for a General Convex Objective**

Now let's consider any convex, twice continuously differentiable objective function $J(\theta)$ that is bounded from below. Suppose that the largest eigenvalue of the Hessian matrix $H = \nabla^2 J(\theta)$ is less than or equal to $\beta_{\max}$ for all points θ.

Show that when running gradient descent, choosing a step size $0 < \alpha < \frac{2}{\beta_{\max}}$ guarantees that $J(\theta^{[t]})$ will converge to some finite value as t approaches infinity.[1] Specifically, you should derive an inequality for $J(\theta^{[t]})$ in terms of $J(\theta^{[t-1]})$, $\nabla J(\theta^{[t-1]})$, α, and $\beta_{\max}$, and show that the objective function is strictly decreasing during each iteration, i.e. $J(\theta^{[t]}) < J(\theta^{[t-1]})$.

Hint: You can assume that, by Taylor's theorem, the following statement is true:

$$J(y) = J(x) + \nabla J(x)^\top (y - x) + \frac{1}{2}(y - x)^\top \nabla^2 J(x + c(y - x))(y - x) \text{ for some } 0 \leq c \leq 1$$

Hint: The Hessian of a convex function is symmetric and positive semi-definite at any point θ, which means all of its eigenvalues are real and non-negative.

Hint: The Hessian matrix is symmetric. Consider using the spectral theorem introduced in problem 3 in homework 0.

Optional (no credit): Using the gradient descent inequality that you derived, show that the Gradient Descent algorithm converges in the sense that $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$.[2]

Remark: This question suggests that a smaller learning rate should be applied if we believe the curvature of the objective function is big.

(e) **[2 points] Learning Rate for Linear Regression**

Consider using Gradient Descent on the Least Mean Square (LMS) objective introduced in lecture:

$$J(\theta) = \frac{1}{2}||X\theta - y||_2^2$$

where $X \in \mathbb{R}^{n\times d}$ is the design matrix of our data, $\theta \in \mathbb{R}^d$ is our parameter, and $\vec{y} \in \mathbb{R}^n$ is the response variable.

Let $\beta_{\max}$ be the largest eigenvalue of $X^\top X$. Prove that for all $\alpha \in (0, \frac{2}{\beta_{\max}})$, Gradient Descent with learning rate α satisfies that $J(\theta^{[t]})$ converges as $t \to \infty$.

Hint: You can invoke the statements in the part (d) (including the optional question in part (d)) to solve this part (even if you were not able to prove part (d).)

Remark: The conclusion here suggests that for linear regression, roughly speaking, you should use smaller learning rates if the scale of your data X is big or if the data points are correlated with each other, both of which will enable a large top eigenvalue of $XX^\top$.

Remark: Even though in many cases the Least Mean Square (LMS) objective can be solved exactly by solving the normal equation as shown in lecture, it is still useful to be able to guarantee convergence when using the Gradient Descent algorithm in situations where it is difficult to solve for the optimal θ^* directly (such as having large amount of data making inverting X very expensive).

---

[1] This definition of convergence is different from the definition in previous questions where we explicitly prove that the parameter θ converges to an optimal parameter θ^*. In this case, we do not know the optimal value θ^*, and it would be difficult to prove convergence using the previous definition.

[2] Note that $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ does not necessarily mean that there exists a vector $\hat{\theta}$ such that $\theta^{[t]} \to \hat{\theta}$. (Even an 1-dimensional counterexample exists.) However, for convex functions, when $\theta^{[t]}$ stays in a bounded set, $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ does imply that $J(\theta^{[t]}) \to \inf_\theta J(\theta)$ as $t \to \infty$. Therefore, $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ is typically considered a reasonable definition for an optimization algorithm's convergence.

2. **[25 points] Locally weighted linear regression**

(a) [10 points] Consider a linear regression problem in which we want to "weight" different training examples differently. Specifically, suppose we want to minimize

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n} w^{(i)} \left(\theta^T x^{(i)} - y^{(i)}\right)^2.$$

In class, we worked out what happens for the case where all the weights (the $w^{(i)}$'s) are the same. In this problem, we will generalize some of those ideas to the weighted setting. We will assume $w^{(i)} > 0$ for all i.

i. [2 points] Show that $J(\theta)$ can also be written

$$J(\theta) = (X\theta - y)^T W (X\theta - y)$$

for an appropriate matrix W, and where X and y are as defined in class. Clearly specify the value of each element of the matrix W.

ii. [4 points] If all the $w^{(i)}$'s equal 1, then we saw in class that the normal equation is

$$X^T X\theta = X^T y,$$

and that the value of θ that minimizes $J(\theta)$ is given by $(X^T X)^{-1} X^T y$. By finding the derivative $\nabla_\theta J(\theta)$ and setting that to zero, generalize the normal equation to this weighted setting, and give the new value of θ that minimizes $J(\theta)$ in closed form as a function of X, W and y.

iii. [4 points] Suppose we have a dataset $\{(x^{(i)}, y^{(i)}); i = 1\ldots, n\}$ of n independent examples, but we model the $y^{(i)}$'s as drawn from conditional distributions with different levels of variance $(\sigma^{(i)})^2$. Specifically, assume the model

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)$$

That is, each $y^{(i)}$ is drawn from a Gaussian distribution with mean $\theta^T x^{(i)}$ and variance $(\sigma^{(i)})^2$ (where the $\sigma^{(i)}$'s are fixed, known, constants). Show that finding the maximum likelihood estimate of θ reduces to solving a weighted linear regression problem. State clearly what the $w^{(i)}$'s are in terms of the $\sigma^{(i)}$'s.
In other words, this suggests that if we have prior knowledge on the noise levels (the variance of the label $y^{(i)}$ conditioned on $x^{(i)}$) of all the examples, then we should use weighted least squares with weights depending on the variances.

(b) [10 points] **Coding problem.**

We will now consider the following dataset (the formatting matches that of Datasets 1-4, except $x^{(i)}$ is 1-dimensional):

`src/lwr/{train,valid,test}.csv`

In `src/lwr/lwr.py`, implement locally weighted linear regression using the normal equations you derived in Part (a) and using

$$w^{(i)} = \exp\left(-\frac{\|x^{(i)} - x\|_2^2}{2\tau^2}\right).$$

Train your model on the `train` split using $\tau = 0.5$, then run your model on the `valid` split and report the Mean Squared Error (MSE). Finally plot your model's predictions on the validation set (plot the training set with blue 'x' markers and the predictions on the validation set with a red 'o' markers). Does the model seem to be under- or overfitting?

(c) [5 points] **Coding problem.**

We will now tune the hyperparameter τ. In `src/lwr/tau.py`, find the Mean Squared Error (MSE) value of your model on the validation set for each of the values of τ specified in the code. For each τ, plot your model's predictions on the validation set in the format described in part (b). Report the value of τ which achieves the lowest MSE on the `valid` split, and finally report the MSE on the `test` split using this τ-value.

3. **[25 points] Linear regression: linear in what?**

In the first two lectures, you have seen how to fit a linear function of the data for the regression problem. In this question, we will see how linear regression can be used to fit non-linear functions of the data using feature maps. We will also explore some of its limitations, for which future lectures will discuss fixes.

(a) [5 points] **Learning degree-3 polynomials of the input**

Suppose we have a dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ where $x^{(i)}, y^{(i)} \in \mathbb{R}$. We would like to fit a third degree polynomial $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the dataset. The key observation here is that the function $h_\theta(x)$ is still linear in the unknown parameter θ, even though it's not linear in the input x. This allows us to convert the problem into a linear regression problem as follows.

Let $\phi : \mathbb{R} \to \mathbb{R}^4$ be a function that transforms the original input x to a 4-dimensional vector defined as

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \in \mathbb{R}^4 \tag{1}$$

Let $\hat{x} \in \mathbb{R}^4$ be a shorthand for $\phi(x)$, and let $\hat{x}^{(i)} \triangleq \phi(x^{(i)})$ be the transformed input in the training dataset. We construct a new dataset $\{(\phi(x^{(i)}), y^{(i)})\}_{i=1}^n = \{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$ by replacing the original inputs $x^{(i)}$'s by $\hat{x}^{(i)}$'s. We see that fitting $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the old dataset is equivalent to fitting a linear function $h_\theta(\hat{x}) = \theta_3 \hat{x}_3 + \theta_2 \hat{x}_2 + \theta_1 \hat{x}_1 + \theta_0$ to the new dataset because

$$h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 = \theta_3 \phi(x)_3 + \theta_2 \phi(x)_2 + \theta_1 \phi(x)_1 + \theta_0 = \theta^T \hat{x} \tag{2}$$

In other words, we can use linear regression on the new dataset to find parameters $\theta_0, \ldots, \theta_3$. You will need to provide two things for this question: 1) Please write down the objective function $J(\theta)$ of the linear regression problem on the new dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$. Specifically, write down a square loss function. 2) Also write the update rule of the batch gradient descent algorithm for linear regression on the dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$.

Terminology: In machine learning, ϕ is often called the feature map which maps the original input x to a new set of variables. To distinguish between these two sets of variables, we will call x the input **attributes**, and call $\phi(x)$ the **features**. (Unfortunately, different authors use different terms to describe these two things. In this course, we will do our best to follow the above convention consistently.)

(b) [5 points] **Coding question: degree-3 polynomial regression**

For this sub-question question, we will use the dataset provided in the following files:

`src/featuremaps/{train,valid,test}.csv`

Each file contains two columns: x and y. In the terminology described in the introduction, x is the attribute (in this case one dimensional) and y is the output label.

Using the formulation of the previous sub-question, implement linear regression with **normal equations** using the feature map of degree-3 polynomials. Use the starter code provided in `src/featuremaps/featuremap.py` to implement the algorithm.

Create a scatter plot of the training data, and plot the learnt hypothesis as a smooth curve over it. Submit the plot in the writeup as the solution for this problem.

Remark: Suppose $\widehat{X}$ is the design matrix of the transformed dataset. You may sometimes encounter a non-invertible matrix $\widehat{X}^T\widehat{X}$. For a numerically stable code implementation, always use `np.linalg.solve` to obtain the parameters directly, rather than explicitly calculating the inverse and then multiplying it with $\widehat{X}^T y$.

(c) [5 points] **Coding question: degree-k polynomial regression**

Now we extend the idea above to degree-k polynomials by considering $\phi : \mathbb{R} \to \mathbb{R}^{k+1}$ to be

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{3}$$

Follow the same procedure as the previous sub-question, and implement the algorithm with $k = 3, 5, 10, 20$. Create a similar plot as in the previous sub-question, and include the hypothesis curves for each value of k with a different color. Include a legend in the plot to indicate which color is for which value of k.

Submit the plot in the writeup as the solution for this sub-problem. Observe how the fitting of the training dataset changes as k increases. Briefly comment on your observations in the plot.

(d) [5 points] **Coding question: other feature maps**

You may have observed that it requires a relatively high degree k to fit the given training data, and this is because the dataset cannot be explained (i.e., approximated) very well by low-degree polynomials. By visualizing the data, you may have realized that y can be approximated well by a sine wave. In fact, we generated the data by sampling from $y = \sin(x) + \xi$, where ξ is noise with Gaussian distribution. Please update the feature map ϕ to include a sine transformation as follows:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \\ \sin(x) \end{bmatrix} \in \mathbb{R}^{k+2} \tag{4}$$

With the updated feature map, train different models for values of $k = 0, 1, 2, 3, 5, 10, 20$, and plot the resulting hypothesis curves over the data as before.

Submit the plot as a solution to this sub-problem. Compare the fitted models with the previous sub-question, and briefly comment about noticeable differences in the fit with this feature map.

(e) [5 points] **Overfitting with expressive models and small data**

For the rest of the problem, we will consider a small dataset (a random subset of the dataset you have been using so far) with much fewer examples, provided in the following file:

`src/featuremaps/small.csv`

We will be exploring what happens when the number of features start becoming bigger than the number of examples in the training set. Run your algorithm on this small dataset using the following feature map

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{5}$$

with $k = 1, 2, 5, 10, 20$.

Create a plot of the various hypothesis curves (just like previous sub-questions). Observe how the fitting of the training dataset changes as k increases. Submit the plot in the writeup and comment on what you observe.

Remark: The phenomenon you observe where the models start to fit the training dataset very well, but suddenly "goes wild" is due to what is called *overfitting*. The intuition to have for now is that, when the amount of data you have is small relative to the expressive capacity of the family of possible models (that is, the hypothesis class, which, in this case, is the family of all degree k polynomials), it results in overfitting.

Loosely speaking, the set of hypothesis function is "very flexible" and can be easily forced to pass through all your data points especially in unnatural ways. In other words, the model explains the noises in the training dataset, which shouldn't be explained in the first place. This hurts the predictive power of the model on test examples. We will describe overfitting in more detail in future lectures when we cover learning theory and bias-variance tradeoffs.

4. **[40 points] Linear Classifiers (logistic regression and GDA)**

In this problem, we cover two probabilistic linear classifiers we have covered in class so far. First, a discriminative linear classifier: logistic regression. Second, a generative linear classifier: Gaussian Discriminant Analysis (GDA). Both of the algorithms find a linear decision boundary that separates the data into two classes, but make different assumptions. Our goal in this problem is to get a deeper understanding of the similarities and differences (and, strengths and weaknesses) of these two algorithms.

For this problem, we will consider two datasets, along with starter codes provided in the following files:

- `src/linearclass/ds1_{train,valid}.csv`
- `src/linearclass/ds2_{train,valid}.csv`
- `src/linearclass/logreg.py`
- `src/linearclass/gda.py`

Each file contains n examples, one example $(x^{(i)}, y^{(i)})$ per row. In particular, the i-th row contains columns $x_1^{(i)} \in \mathbb{R}$, $x_2^{(i)} \in \mathbb{R}$, and $y^{(i)} \in \{0, 1\}$. In the subproblems that follow, we will investigate using logistic regression and Gaussian discriminant analysis (GDA) to perform binary classification on these two datasets.

(a) [10 points]
In lecture we saw the average empirical loss for logistic regression:

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right),$$

where $y^{(i)} \in \{0, 1\}$, $h_\theta(x) = g(\theta^T x)$ and $g(z) = 1/(1 + e^{-z})$.
Find the Hessian H of this function, and show that for any vector z, it holds true that

$$z^T H z \geq 0.$$

Hint: You may want to start by showing that $\sum_i \sum_j z_i x_i x_j z_j = (x^T z)^2 \geq 0$. Recall also that $g'(z) = g(z)(1 - g(z))$.
Remark: This is one of the standard ways of showing that the matrix H is positive semi-definite, written "$H \succeq 0$." This implies that J is convex, and has no local minima other than the global one. If you have some other way of showing $H \succeq 0$, you're also welcome to use your method instead of the one above.

(b) [5 points] **Coding problem.** Follow the instructions in `src/linearclass/logreg.py` to train a logistic regression classifier using Newton's Method. Starting with $\theta = \vec{0}$, run Newton's Method until the updates to θ are small: Specifically, train until the first iteration k such that $\|\theta_k - \theta_{k-1}\|_1 < \epsilon$, where $\epsilon = 1 \times 10^{-5}$. Make sure to **write your model's predicted probabilities** on the validation set to the file specified in the code.
Include a plot of the **validation data** with x_1 on the horizontal axis and x_2 on the vertical axis. To visualize the two classes, use a different symbol for examples $x^{(i)}$ with $y^{(i)} = 0$ than for those with $y^{(i)} = 1$. On the same figure, plot the decision boundary found by logistic regression (i.e., line corresponding to $p(y|x) = 0.5$).

(c) [5 points] Recall that in Gaussian Discriminant Analysis (GDA) we model the joint distribution of (x, y) by the following equations:

$$p(y) = \begin{cases} \phi & \text{if } y = 1 \\ 1 - \phi & \text{if } y = 0 \end{cases}$$
$$p(x|y = 0) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_0)^T \Sigma^{-1}(x - \mu_0)\right)$$
$$p(x|y = 1) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_1)^T \Sigma^{-1}(x - \mu_1)\right),$$

where ϕ, μ_0, μ_1, and Σ are the parameters of our model.

Suppose we have already fit ϕ, μ_0, μ_1, and Σ, and now want to predict y given a new point x. To show that GDA results in a classifier that has a linear decision boundary, show the posterior distribution can be written as

$$p(y = 1 \mid x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-(\theta^T x + \theta_0))},$$

where $\theta \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$ are appropriate functions of ϕ, Σ, μ_0, and μ_1.

(d) [7 points] Given the dataset, we claim that the maximum likelihood estimates of the parameters are given by

$$\phi = \frac{1}{n} \sum_{i=1}^{n} 1\{y^{(i)} = 1\}$$
$$\mu_0 = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{n} 1\{y^{(i)} = 0\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}}$$
$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

The log-likelihood of the data is

$$\begin{aligned} \ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi). \end{aligned}$$

By maximizing ℓ with respect to the four parameters, prove that the maximum likelihood estimates of ϕ, μ_0, μ_1, and Σ are indeed as given in the formulas above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of μ_0 and μ_1 above are non-zero.)

(e) [5 points] **Coding problem.** In `src/linearclass/gda.py`, fill in the code to calculate ϕ, μ_0, μ_1, and Σ, use these parameters to derive θ, and use the resulting Gaussian Discriminant Analysis (GDA) model to make predictions on the validation set. Make sure to write your model's predictions on the validation set to the file specified in the code.

Include a plot of the **validation data** with x_1 on the horizontal axis and x_2 on the vertical axis. To visualize the two classes, use a different symbol for examples $x^{(i)}$ with $y^{(i)} = 0$ than for those with $y^{(i)} = 1$. On the same figure, plot the decision boundary found by GDA (i.e, line corresponding to $p(y|x) = 0.5$).

(f) [2 points] For Dataset 1, compare the validation set plots obtained in part (b) and part (e) from logistic regression and GDA respectively, and briefly comment on your observation in a couple of lines.

(g) [5 points] Repeat the steps in part (b) and part (e) for Dataset 2. Create similar plots on the **validation set** of Dataset 2 and include those plots in your writeup.

On which dataset does GDA seem to perform worse than logistic regression? Why might this be the case?

(h) **[1 points]** For the dataset where GDA performed worse in parts (f) and (g), can you find a transformation of the $x^{(i)}$'s such that GDA performs significantly better? What might this transformation be?

5. **[12 points] Logistic Regression: Training stability**

In this problem, we will be delving deeper into the workings of logistic regression. The goal of this problem is to help you develop your skills debugging machine learning algorithms (which can be very different from debugging software in general).

We have provided an implementation of logistic regression in `src/stability/stability.py`, and two labeled datasets A and B in `src/stability/ds1_a.csv` and `src/stability/ds1_b.csv`.

Please do not modify the code for the logistic regression training algorithm for this problem. First, run the given logistic regression code to train two different models on A and B. You can run the code by simply executing `python stability.py` in the `src/stability` directory.

(a) [2 points] What is the most notable difference in training the logistic regression model on datasets A and B?

(b) [5 points] Investigate why the training procedure behaves unexpectedly on dataset B, but not on A. Provide hard evidence (in the form of math, code, plots, etc.) to corroborate your hypothesis for the misbehavior. Remember, you should address why your explanation does *not* apply to A.

Hint: The issue is not a numerical rounding or over/underflow error.

(c) [5 points] For each of these possible modifications, state whether or not it would lead to the provided training algorithm converging on datasets such as B. Justify your answers.

i. Using a different constant learning rate.

ii. Decreasing the learning rate over time (e.g. scaling the initial learning rate by $1/t^2$, where t is the number of gradient descent iterations thus far).

iii. Linear scaling of the input features.

iv. Adding a regularization term $\|\theta\|_2^2$ to the loss function.

v. Adding zero-mean Gaussian noise to the training data or labels.