CS 229, Fall 2024
Problem Set #2

YOUR NAME HERE (`YOUR SUNET HERE`)

Due Wednesday, October 23 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/67631/discussion/`.

(3) 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) The due date is Wednesday, October 23 at 11:59 pm. If you submit after Wednesday, October 23 at 11:59 pm, you will begin consuming your late days. The late day policy can be found in the course website: Course Logistics and FAQ.

All students must submit an electronic PDF version of the written question including plots generated from the codes. We highly recommend typesetting your solutions via LaTeX. 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. Please make sure that your PDF file and zip file are submitted to the corresponding Gradescope assignments respectively. We reserve the right to not give any points to the written solutions if the associated code is not submitted.

Honor code: We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solution independently, and without referring to written notes from the joint session. Each student must understand the solution well enough in order to reconstruct it by him/herself. It is an honor code violation to copy, refer to, or look at written or code solutions from a previous year, including but not limited to: official solutions from a previous year, solutions posted online, and solutions you or someone else may have written up in a previous year. Furthermore, it is an honor code violation to post your assignment solutions online, such as on a public git repo. We run plagiarism-detection software on your code against past solutions as well as student submissions from previous years. Please take the time to familiarize yourself with the Stanford Honor Code[1] and the Stanford Honor Code[2] as it pertains to CS courses.

[1]https://communitystandards.stanford.edu/policies-and-guidance/honor-code
[2]https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1164/handouts/honor-code.pdf

1. **[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?

Answer: The same code converges when trained on dataset A, but does not converge when trained on dataset 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.

Answer: The reason is dataset B is linearly separable while A is not. Logistic regression without regularization (as implemented in the code) will not converge to a single parameter if the data is perfectly separable (similar to X^TX being singular in linear regression). Since data is perfectly separable, any θ that separates the data can be increased in magnitude to further decrease the loss, without change in the decision boundary.

If left to run, $\|\theta\|_2$ will slowly, but surely increase forever (towards infinity).

(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.

Answer:

i. No.
ii. No / may help. Eventually updates will become arbitrarily small. Still, the norm of theta can end up being very big. You may also end up underfitting if you aggressively decrease the learning rate.
iii. No. This will not affect whether the examples are linearly separable.
iv. Yes. This penalizes large norm of theta and makes the training converge on all kinds of data.
v. No / may help if the noise happens to make the data linearly inseparable.

2. **[20 points] Decision Trees and Gini Loss**

When growing a decision tree, we split the input space in a greedy, top-down, recursive manner. Given a parent region R_p, we can choose a split $s_p(j,t)$ which yields two child regions $R_1 = \{X \mid x_j < t, X \in R_p\}$ and $R_2 = \{X \mid x_j \geq t, X \in R_p\}$. Assuming we have defined a per region loss $L(R)$, at each branch we select the split that minimizes the weighted loss of the children:

$$\min_{j,t} \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|}$$

When performing classification, a commonly used loss is the Gini loss, defined for the K-class classification problem as:

$$G(R_m) = G(\vec{p}_m) = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$$

Where $\vec{p}_m = \begin{bmatrix} p_{m1} & p_{m2} & \dots & p_{mK} \end{bmatrix}$ and p_{mk} is the proportion of examples of class k that are are present in region R_m. However, we are oftentimes more interested in optimizing the final misclassification loss:

$$M(R_m) = M(\vec{p}_m) = 1 - \max_k p_{mk} \tag{1}$$

For the problems below, assume we are dealing with binary classification and that there are no degenerate cases where positive and negative datapoints overlap in the feature space.

(a) [5 points] Show that for any given split, the weighted Gini loss of the children can not exceed that of the parent. (**Hint**: first show that the Gini loss is strictly concave. And then use the fact that G is strictly concave meaning:

$$\forall p_1 \neq p_2, \forall t \in (0,1) : G(tp_1 + (1-t)p_2) > tG(p_1) + (1-t)G(p_2)$$

Answer: First note that we can re-write the Gini loss in the binary class in terms of a single proportion p_m, which we define as the proportion of positive examples in the region R_m:

$$\begin{aligned} G(\vec{p}_m) = G(p_m) &= p_m(1-p_m) + (1-p_m)(1-(1-p_m)) \\ &= 2p_m - 2p_m^2 \end{aligned}$$

Thus, $\frac{d^2G}{dp_m^2}(p_m) = -4$ and the Gini loss is strongly concave.
Defining $t = \frac{|R_1|}{|R_1|+|R_2|}$, we can write the parent region R_p proportion p_p as:

$$p_p = t * p_1 + (1-t) * p_2$$

Thus, we can use the definition of concavity to show that:

$$\begin{aligned} G(R_p) &= G(p_p) \\ &= G(tp_1 + (1-t)p_2) \\ &\geq tG(p_1) + (1-t)G(p_2) \\ &= \frac{|R_1|G(R_1) + |R_2|G(R_2)}{|R_1| + |R_2|} \end{aligned}$$

(b) [5 points] List out the cases where Gini loss will stay the same after a split. Show why these do not violate the strong concavity of the Gini loss. Briefly explain why these cases do not prevent a fully grown tree from achieving zero Gini loss. (**Hint**: Recall the definition of strict concavity).

Answer: There are two cases where the loss of the children will not be lower than the loss of the parent. The first is when one of the two children are empty. Then either $t = 0$ or $t = 1$, which is outside the range where the inequality is forced to be strict. The second case is when the proportion of positive and negative examples in each child remains the same as that of the parent. Then we have that $p_p = p_1 = p_2$, which again allows the inequality to not be strict.

These cases do not prevent a tree from fully fitting the data, as the lack of degenerate examples will always allow us to pick splits that have non-zero cardinality children, and the constant diminishing of the cardinality of the children will eventually force an uneven split where $p_1 \neq p_2$.

(c) [4 points] If instead we use misclassification loss, what additional case causes the loss to stay the same after a split? Show why this is (hint: you may find it useful to define $N_m = |R_m|$ and N_{mk} as the number of examples of class k present in R_m).

Answer: If the majority class of the two children match that of the parent, the misclassification loss will remain the same. To show this, first define class c as the majority class prediction. We then have that:

$$\begin{aligned}
\frac{|R_1|M(R_1) + |R_2|M(R_2)}{|R_1| + |R_2|} &= \frac{N_1(1 - p_{1c}) + N_2(1 - p_{2c})}{N_1 + N_2} \\
&= \frac{N_1(1 - \frac{N_{1c}}{N_1}) + N_2(1 - \frac{N_{2c}}{N_2})}{N_1 + N_2} \\
&= 1 - \frac{N_{1c} + N_{2c}}{N_1 + N_2} \\
&= 1 - \frac{N_{pc}}{N_p} = M(R_p)
\end{aligned}$$

(d) [4 points] Consider a training set X. In bootstrap sampling, each time we draw a random sample Z of size N from the training data and obtain $Z_1, Z_2, ..., Z_B$ after B times, i.e. we generate B different bootstrapped training data sets. If we apply bagging to regression trees, each time a tree $T_i(i = 1, 2, ..., B)$ is grown based on the bootstrapped data Z_i, and we average all the predictions to get:

$$\hat{T(x)} = \frac{1}{B}\sum_{i=1}^{B} T_i(x)$$

Now, if $T_1, T_2, ..., T_B$ is independent from each other, but each has the same variance σ^2, the variance of the average $\hat{T}$ is σ^2/B. However, in practice, the bagged trees could be similar to each other, resulting in correlated predictions. Assume $T_1, T_2, ..., T_B$ still share the same variance σ^2, but have a positive pair-wise correlation ρ. We define the correlation between two random variables as:

$$Corr(X, Y) = \frac{Cov(X, Y)}{\sqrt{Var(X)}\sqrt{Var(Y)}}$$

Thus, we have $\rho = Corr(T_i(x), T_j(x)), i \neq j$.
Show that in this case, the variance of the average is given by:

$$Var(\frac{1}{B}\sum_{i=1}^{B} T_i(x)) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Answer:

$$\begin{aligned} Var(\frac{1}{B}\sum_{i=1}^{B} T_i(x)) &= \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B} Cov(T_i(x), T_j(x)) \\ &= \frac{1}{B^2}\sum_{i=1}^{B}(Cov(T_i(x), T_i(x)) + \sum_{j\neq i}^{B} Cov(T_i(x), T_j(x))) \\ &= \frac{1}{B^2}\sum_{i=1}^{B}(\sigma^2 + (B-1)\sigma^2\rho) \\ &= \frac{B(\sigma^2 + (B-1)\sigma^2\rho)}{B^2} \\ &= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \end{aligned}$$

3. **[20 points] AdaBoost Performance**

We learned about boosting in lecture. Statistician Kevin Murphy claims that "It can be shown that, as long as each base learner has an accuracy that is better than chance (even on the weighted dataset), then the final ensemble of classifiers will have higher accuracy than any given component." We will now verify this in the AdaBoost framework.

(a) [3 points] Given a set of n observations (x_i, y_i) where y_i is the label $y_i \in \{-1, 1\}$, let $f_t(x)$ be the weak classifier at step t and let $\hat{w}_t$ be its weight. First we note that the final classifier after T steps is defined as

$$F(x) = \text{sign}\left\{\sum_{t=1}^{T} \hat{w}_t f_t(x)\right\} = \text{sign}\{f(x)\},$$

where

$$f(x) = \sum_{t=1}^{T} \hat{w}_t f_t(x).$$

We can assume that $f(x)$ is never exactly zero.
Show that

$$\varepsilon_{\text{training}} := \frac{1}{n}\sum_{i=1}^{n} 1_{\{F(x_i)\neq y_i\}} \leq \frac{1}{n}\sum_{i=1}^{n} \exp(-f(x_i)y_i),$$

where $1_{\{F(x_i)\neq y_i\}}$ is 1 if $F(x_i) \neq y_i$ and 0 otherwise.
Answer: It suffices to show that for any i,

$$1_{\{F(x_i)\neq y_i\}} \leq \exp(-f(x_i)y_i). \tag{2}$$

In fact, if $F(x_i) \neq y_i$, then the sign of $f(x_i)$ and y_i are different, and thus $f(x_i)y_i < 0$. This means that $\exp(-f(x_i)y_i) > 1 = 1_{\{F(x_i)\neq y_i\}}$. Similarly, we can show Eq. (2) is true if $F(x_i) = y_i$.

(b) [8 points] The weight for each data point i at step $t+1$ can be defined recursively by

$$\alpha_{i,(t+1)} = \frac{\alpha_{i,t}\exp(-\hat{w}_t f_t(x_i)y_i)}{Z_t},$$

where Z_t is a normalizing constant ensuring the weights sum to 1

$$Z_t = \sum_{i=1}^{n} \alpha_{i,t}\exp(-\hat{w}_t f_t(x_i)y_i).$$

Show that

$$\frac{1}{n}\sum_{i=1}^{n}\exp(-f(x_i)y_i) = \prod_{t=1}^{T} Z_t.$$

Answer: Notice that

$$\frac{\alpha_{i,(t+1)}}{\alpha_{i,t}} = \frac{\exp(-\hat{w}_t f_t(x_i)y_i)}{Z_t},$$

then

$$\frac{\alpha_{i,(T+1)}}{\alpha_{i,1}} = \prod_{t=1}^{T} \frac{\alpha_{i,(t+1)}}{\alpha_{i,t}} = \prod_{t=1}^{T} \frac{\exp(-\hat{w}_t f_t(x_i) y_i)}{Z_t} = \frac{\exp(-y_i \sum_{t=1}^{T} \hat{w}_t f_t(x_i))}{\prod_{t=1}^{T} Z_t} = \frac{\exp(-y_i f(x_i))}{\prod_{t=1}^{T} Z_t}.$$

Since $\alpha_{i,1} = 1/n$ and $\sum_{i=1}^{n} \alpha_{i,(T+1)} = 1$, we have

$$\sum_{i=1}^{n} \frac{\exp(-y_i f(x_i))}{\prod_{t=1}^{T} Z_t} = \sum_{i=1}^{n} \frac{\alpha_{i,(T+1)}}{\alpha_{i,1}} = n \sum_{i=1}^{n} \alpha_{i,(T+1)} = n.$$

(c) [9 points] We showed above that training error is bounded above by $\prod_{t=1}^{T} Z_t$. At step t the values $Z_1, Z_2, \ldots, Z_{t-1}$ are already fixed therefore at step t we can choose α_t to minimize Z_t. Let

$$\varepsilon_t = \sum_{i=1}^{n} \alpha_{i,t} 1_{\{f_t(x_i) \neq y_i\}}$$

be the weighted training error for the weak classifier $f_t(x)$. Then we can re-write the formula for Z_t as

$$Z_t = (1 - \varepsilon_t) \exp(-\hat{w}_t) + \varepsilon_t \exp(\hat{w}_t).$$

(i) [3 points] First find the value of $\hat{w}_t$ that minimizes Z_t. Then show that the corresponding optimal value is

$$Z_t^{\text{opt}} = 2\sqrt{\varepsilon_t(1 - \varepsilon_t)}.$$

(ii) [3 points] Assume we choose Z_t this way. Then re-write $\varepsilon_t = 1/2 - \gamma_t$, where $\gamma_t > 0$ implies better than random and $\gamma_t < 0$ implies worse than random. Then show that

$$Z_t \leq \exp(-2\gamma_t^2).$$

(You may want to use the fact that $\log(1 - x) \leq -x$ for $0 \leq x < 1$.)

(iii) [3 points] Finally, show that if each classifier is better than random, i.e., $\gamma_t > \gamma$ for all t and $\gamma > 0$, then

$$\varepsilon_{\text{training}} \leq \exp(-2T\gamma^2),$$

which shows that the training error can be made arbitrarily small with enough steps.

Answer:

(i) We take the partial derivative of Z_t with regard to $\hat{w}_t$,

$$\frac{\partial Z_t}{\partial \hat{w}_t} = \varepsilon_t \exp(\hat{w}_t) - (1 - \varepsilon_t) \exp(-\hat{w}_t).$$

Set the partial derivative to zero and we obtain

$$\hat{w}_t^{\text{opt}} = \frac{1}{2} \log\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right).$$

Plug the above $\hat{w}_t^{\text{opt}}$ in Z_t,

$$Z_t^{\text{opt}} = (1 - \varepsilon_t)\sqrt{\frac{\varepsilon_t}{1 - \varepsilon_t}} + \varepsilon_t\sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}} = 2\sqrt{\varepsilon_t(1 - \varepsilon_t)}.$$

(ii) Plug $\varepsilon_t = 1/2 - \gamma_t$ into the formula of Z_t^{opt},

$$Z_t^{\mathsf{opt}} = \sqrt{(1 - 2\gamma_t)(1 + 2\gamma_t)} = \sqrt{1 - 4\gamma_t^2}.$$

Since $1 - x \le e^{-x}$ for $0 \le x < 1$,

$$\sqrt{1 - 4\gamma_t^2} \le \sqrt{\exp(-4\gamma_t^2)} = \exp(-2\gamma_t^2),$$

and we finish the proof.

(iii) Combine (a), (b), and (c)(ii),

$$\varepsilon_{\mathsf{training}} \le \prod_{t=1}^{T} Z_t \le \exp\left(-2\sum_{t=1}^{T} \gamma_t^2\right).$$

If $\gamma_t > \gamma$ for all t and $\gamma > 0$, then

$$\varepsilon_{\mathsf{training}} \le \exp\left(-2T\gamma^2\right).$$

4. **[20 points] Spam classification**

In this problem, we will use the naive Bayes algorithm to build a spam classifier.

In recent years, spam on electronic media has been a growing concern. Here, we'll build a classifier to distinguish between real messages, and spam messages. For this class, we will be building a classifier to detect SMS spam messages. We will be using an SMS spam dataset developed by Tiago A. Almedia and José María Gómez Hidalgo which is publicly available on `http://www.dt.fee.unicamp.br/~tiago/smsspamcollection` [3]

We have split this dataset into training and testing sets and have included them in this assignment as `src/spam/spam_train.tsv` and `src/spam/spam_test.tsv`. See `src/spam/spam_readme.txt` for more details about this dataset. Please refrain from redistributing these dataset files. The goal of this assignment is to build a classifier from scratch that can tell the difference the spam and non-spam messages using the text of the SMS message.

(a) [5 points] Implement code for processing the the spam messages into numpy arrays that can be fed into machine learning models. Do this by completing the `get_words`, `create_dictionary`, and `transform_text` functions within our provided `src/spam.py`. Do note the corresponding comments for each function for instructions on what specific processing is required.

The provided code will then run your functions and save the resulting dictionary into `spam_dictionary` and a sample of the resulting training matrix into `spam_sample_train_matrix`. In your writeup, report the vocabular size after the pre-processing step. You do not need to include any other output for this subquestion.

Answer: Size of dictionary: 1722 if using `.split(' ')`, or 1721 if using `.split()`. 1717 if using `.split('|')`

(b) [10 points] In this question you are going to implement a naive Bayes classifier for spam classification with **multinomial event model** and Laplace smoothing.

Code your implementation by completing the `fit_naive_bayes_model` and `predict_from_naive_bayes_model` functions in `src/spam/spam.py`.

Now `src/spam/spam.py` should be able to train a Naive Bayes model, compute your prediction accuracy and then save your resulting predictions to `spam_naive_bayes_predictions`.

In your writeup, report the accuracy of the trained model on the **test set**.

Remark. If you implement naive Bayes the straightforward way, you will find that the computed $p(x|y) = \prod_i p(x_i|y)$ often equals zero. This is because $p(x|y)$, which is the product of many numbers less than one, is a very small number. The standard computer representation of real numbers cannot handle numbers that are too small, and instead rounds them off to zero. (This is called "underflow.") You'll have to find a way to compute Naive Bayes' predicted class labels without explicitly representing very small numbers such as $p(x|y)$. [**Hint:** Think about using logarithms.]

Answer: Naive Bayes had an accuracy of 0.978494623655914 on the testing set.

(c) [5 points] Intuitively, some tokens may be particularly indicative of an SMS being in a particular class. We can try to get an informal sense of how indicative token i is for the SPAM class by looking at:

$$\log \frac{p(x_j = i \mid y = 1)}{p(x_j = i \mid y = 0)} = \log \left(\frac{P(\text{token } i \mid \text{email is SPAM})}{P(\text{token } i \mid \text{email is NOTSPAM})} \right).$$

[3]Almeida, T.A., Gómez Hidalgo, J.M., Yamakami, A. Contributions to the Study of SMS Spam Filtering: New Collection and Results. Proceedings of the 2011 ACM Symposium on Document Engineering (DOCENG'11), Mountain View, CA, USA, 2011.

Complete the `get_top_five_naive_bayes_words` function within the provided code using the above formula in order to obtain the 5 most indicative tokens. Report the top five words in your writeup.

Answer: The top 5 indicative words for Naive Bayes are: ['claim', 'won', 'prize', 'tone', 'urgent!'].

5. **[25 points] Linear Classifiers (logistic regression and GDA) Continued**

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.

Typically, a trained model is evaluated by its performance on the validation dataset. The validation dataset is a set of examples drawn from the same (or a similar) distribution as the training data. Intuitively, this is because we need the trained model to correctly predict the label for not only the training data, but also new samples from the same distribution.

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

$$p(y) = \begin{cases} \phi & \text{if } y = 1 \\ 1 - \phi & \text{if } y = 0 \end{cases} \tag{3}$$

$$p(x|y=0) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right) \tag{4}$$

$$p(x|y=1) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right),$$

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. State the value of θ and θ_0 as a function of $\phi, \mu_0, \mu_1, \Sigma$ explicitly.

Answer: For shorthand, we let $\mathcal{H} = \{\phi, \Sigma, \mu_0, \mu_1\}$ denote the parameters for the problem. Since the given formulae are conditioned on y, use Bayes rule to get:

$$\begin{aligned}
&p(y = 1|x; \phi,\Sigma, \mu_0, \mu_1)\\
&\quad = \frac{p(x|y = 1; \phi, \Sigma, \mu_0, \mu_1)p(y = 1; \phi, \Sigma, \mu_0, \mu_1)}{p(x; \phi, \Sigma, \mu_0, \mu_1)}\\
&\quad = \frac{p(x|y = 1; \mathcal{H})p(y = 1; \mathcal{H})}{p(x|y = 1; \mathcal{H})p(y = 1; \mathcal{H}) + p(x|y = 0; \mathcal{H})p(y = 0; \mathcal{H})}\\
&\quad = \frac{\exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)\phi}{\exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)\phi + \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)(1-\phi)}\\
&\quad = \frac{1}{1 + \frac{1-\phi}{\phi}\exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) + \frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)}\\
&\quad = \frac{1}{1 + \exp\left(\log(\frac{1-\phi}{\phi}) - \frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) + \frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)}.
\end{aligned}$$

Now, we expand and rearrange the difference of quadratic terms in the preceding expression, finding that

$$\begin{aligned}
&(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) - (x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\\
&= x^T\Sigma^{-1}x - \mu_0^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_0 + \mu_0^T\Sigma^{-1}\mu_0 - x^T\Sigma^{-1}x + \mu_1^T\Sigma^{-1}x + x^T\Sigma^{-1}\mu_1 - \mu_1^T\Sigma^{-1}\mu_1\\
&= -2\mu_0^T\Sigma^{-1}x + \mu_0^T\Sigma^{-1}\mu_0 + 2\mu_1^T\Sigma^{-1}x - \mu_1^T\Sigma^{-1}\mu_1\\
&= 2(\mu_1-\mu_0)^T\Sigma^{-1}x + \mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1.
\end{aligned}$$

Thus, we have

$$p(y = 1 \mid x; \mathcal{H}) = \frac{1}{1 + \exp\left(\log\frac{1-\phi}{\phi} + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 + (\mu_0-\mu_1)^T\Sigma^{-1}x\right)}.$$

and setting

$$\theta = -\Sigma^{-1}(\mu_0 - \mu_1) \text{ and } \theta_0 = \frac{1}{2}(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1) - \log\frac{1-\phi}{\phi}$$

gives that

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

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

$$\phi = \frac{1}{n}\sum_{i=1}^{n} 1\{y^{(i)} = 1\} \tag{5}$$

$$\mu_0 = \frac{\sum_{i=1}^n 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^n 1\{y^{(i)} = 0\}} \tag{6}$$

$$\mu_1 = \frac{\sum_{i=1}^n 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^n 1\{y^{(i)} = 1\}} \tag{7}$$

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

The log-likelihood of the data is

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \tag{8}$$
$$= \log \prod_{i=1}^{n} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi).$$

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.)

Answer:

First, derive the expression for the log-likelihood of the training data:

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{n} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi) \tag{9}$$
$$= \sum_{i=1}^{n} \log p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) + \sum_{i=1}^{n} \log p(y^{(i)}; \phi) \tag{10}$$
$$\simeq \sum_{i=1}^{n} [\frac{1}{2} \log \frac{1}{|\Sigma|} - \frac{1}{2}(x^{(i)} - \mu_{y^{(i)}})^T \Sigma^{-1} (x^{(i)} - \mu_{y^{(i)}}) + \frac{y^{(i)}}{2} \log \phi + \frac{1 - y^{(i)}}{2} \log(1 - \phi)]$$

where constant terms indepedent of the parameters have been ignored in the last expression.

Now, the likelihood is maximized by setting the derivative (or gradient) with respect to each of the parameters to zero.

$$\frac{\partial \ell}{\partial \phi} = \sum_{i=1}^{n} [\frac{y^{(i)}}{2\phi} - \frac{1 - y^{(i)}}{2(1 - \phi)}] \tag{11}$$
$$= \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}}{2\phi} - \frac{n - \sum_{i=1}^{n} 1\{y^{(i)} = 1\}}{2(1 - \phi)}$$

Setting this equal to zero and solving for ϕ gives the maximum likelihood estimate.

For μ_0, take the gradient of the log-likelihood, and then use the same kinds of tricks as were used to analytically solve the linear regression problem.

$$\nabla_{\mu_0} \ell = -\frac{1}{2} \sum_{i:y^{(i)}=0} \nabla_{\mu_0} (x^{(i)} - \mu_0)^T \Sigma^{-1} (x^{(i)} - \mu_0) \tag{12}$$
$$= -\frac{1}{2} \sum_{i:y^{(i)}=0} \nabla_{\mu_0} \left[\mu_0^T \Sigma^{-1} \mu_0 - {x^{(i)}}^T \Sigma^{-1} \mu_0 - \mu_0^T \Sigma^{-1} x^{(i)}\right] \tag{13}$$
$$= -\frac{1}{2} \sum_{i:y^{(i)}=0} \left[2\Sigma^{-1} \mu_0 - 2\Sigma^{-1} x^{(i)}\right]$$

The last step uses matrix calculus identities, and also the fact that Σ (and thus Σ^{-1}) is symmetric.

Setting this gradient to zero gives the maximum likelihood estimate for μ_0. The derivation for μ_1 is similar to the one above.

For Σ, we find the gradient with respect to $S = \Sigma^{-1}$ rather than Σ just to simplify the derivation (note that $|S| = \frac{1}{|\Sigma|}$). You should convince yourself that the maximum likelihood estimate S_n found in this way would correspond to the actual maximum likelihood estimate Σ_n as $S_n^{-1} = \Sigma_n$.

$$\nabla_S \ell = \sum_{i=1}^{n} \nabla_S \big[\frac{1}{2}\log|S| - \frac{1}{2}\underbrace{(x^{(i)} - \mu_{y^{(i)}})^T}_{b_i^T} S \underbrace{(x^{(i)} - \mu_{y^{(i)}})}_{b_i}\big] \tag{14}$$

$$= \sum_{i=1}^{n} \big[\frac{1}{2|S|}\nabla_S|S| - \frac{1}{2}\nabla_S b_i^T S b_i\big] \tag{15}$$

But, we have the following identities:

$$\nabla_S |S| = |S|(S^{-1})^T$$

$$\nabla_S b_i^T S b_i = \nabla_S tr\left(b_i^T S b_i\right) = \nabla_S tr\left(S b_i b_i^T\right) = b_i b_i^T$$

In the above, we again used matrix calculus identities, and also the commutatitivity of the trace operator for square matrices. Putting these into the original equation, we get:

$$\nabla_S \ell = \sum_{i=1}^{n} \big[\frac{1}{2}S^{-1} - \frac{1}{2}b_i b_i^T\big] \tag{16}$$

$$= \frac{1}{2}\sum_{i=1}^{n} \big[\Sigma - b_i b_i^T\big]$$

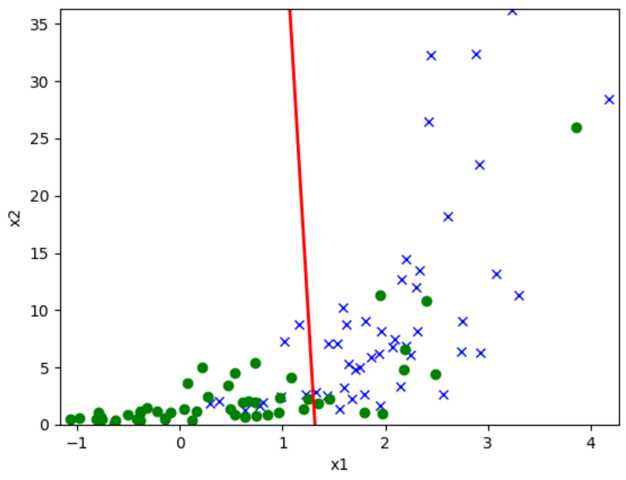
Setting this to zero gives the required maximum likelihood estimate for Σ.

(c) [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 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.

In addition, use the provided code in `src/linearclass/logreg.py` to fit a logistic regression and make predictions on the validation set.

Include two plots 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 $y^{(i)} = 1$. In the first figure, plot the decision boundary found by GDA (i.e, line corresponding to $p(y|x) = 0.5$). In the second figure, plot the decision boundary found by logistic regression.

Answer:



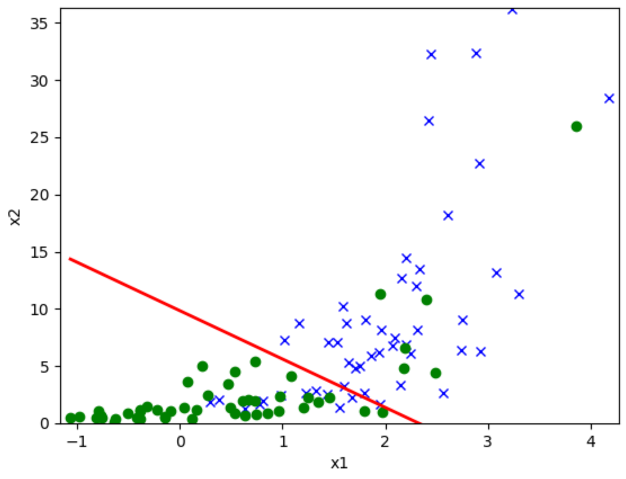


Figure 1: Separating hyperplane for GDA (top) and logistic regression (bottom) on Dataset 1

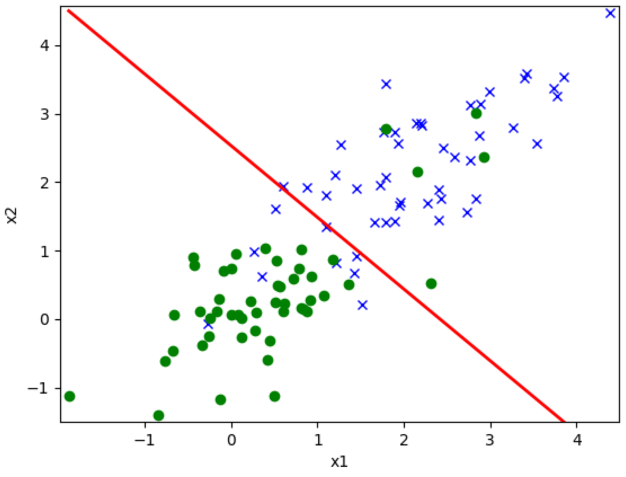
(d) [2 points] For Dataset 1, compare the validation set plots obtained in part (c) from logistic regression and GDA respectively, and briefly comment on your observation in a couple of lines.

Answer: The decision boundary with logistic regression appears more reasonable than GDA on Dataset 1. The accuracy is also higher with the logistic regression model on Dataset 1 (Logistic Regression Accuracy- 0.83, GDA Accuracy - 0.81).

(e) [5 points] Repeat the steps in part (c) 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?

Answer:



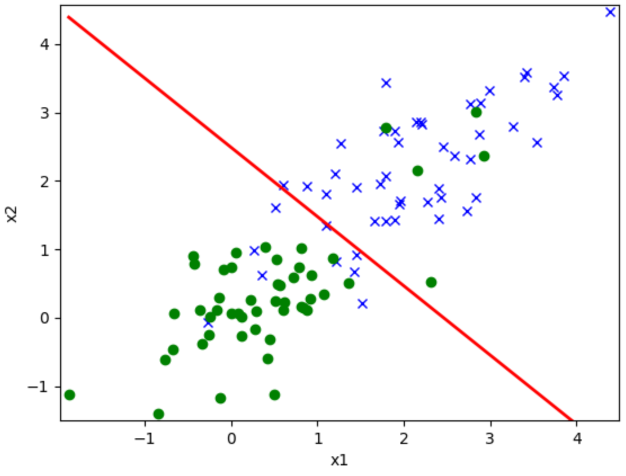


Figure 2: Separating hyperplanes for logistic regression (top) and GDA (bottom) on Dataset 2.

GDA seems to perform worse than logistic regression on Dataset 1 (GDA Accuracy: 0.81 and LR Accuracy: 0.83), while on Dataset 2 they both achieve similar accuracy (GDA/LR Accuracy: 0.86). This is probably because the $x^{(i)}$'s are not Gaussian. We saw in the notes that logistic regression is more robust than GDA when the underlying dataset is not drawn from a multivariate Gaussian.

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

Answer: Full points awarded to any transformation that aim to make Dataset 1 Gaussian, e.g. by setting $x_2 := \log x_2$.