XCS229 Problem Set 2

Due Sunday, April 30 at 11:59pm PT.

Guidelines

1. If you have a question about this homework, we encourage you to post your question on our Slack channel, at `http://xcs229-scpd.slack.com/`
2. Familiarize yourself with the collaboration and honor code policy before starting work.
3. For the coding problems, you must use the packages specified in the provided environment description. Since the autograder uses this environment, we will not be able to grade any submissions which import unexpected libraries.

Submission Instructions

Written Submission: Some questions in this assignment require a written response. For these questions, you should submit a PDF with your solutions online in the online student portal. As long as the PDF is legible and organized, the course staff has no preference between a handwritten and a typeset LaTeX submission. If you wish to typeset your submission and are new to LaTeX, you can get started with the following:

- Type responses only in `submission.tex`.
- Submit the compiled PDF, **not** `submission.tex`.
- Use the commented instructions within the `Makefile` and `README.md` to get started.

Coding Submission: Some questions in this assignment require a coding response. For these questions, you should submit **all files indicated in the question** to the online student portal. For further details, see Writing Code and Running the Autograder below.

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 solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions. More information regarding the Stanford honor code can be foudn at `https://communitystandards.stanford.edu/policies-and-guidance/honor-code`.

Writing Code and Running the Autograder

All your code should be entered into the `src/submission/` directory. When editing files in `src/submission/`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files outside the `src/submission/` directory.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These tests are provided to make sure that your inputs and outputs are on the right track, and that the hidden evaluation tests will be able to execute.

- `hidden`: These unit tests are the evaluated elements of the assignment, and run your code with more complex inputs and corner cases. Just because your code passed the basic local tests does not necessarily mean that they will pass all of the hidden tests. These evaluative hidden tests will be run when you submit your code to the Gradescope autograder via the online student portal, and will provide feedback on how many points you have earned.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

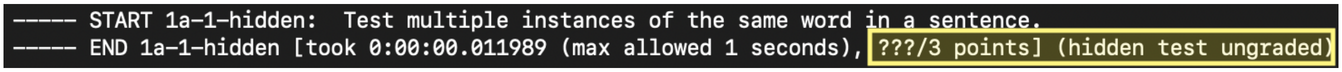
Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

Test Cases

The autograder is a thin wrapper over the python `unittest` framework. It can be run either locally (on your computer) or remotely (on SCPD servers). The following description demonstrates what test results will look like for both local and remote execution. For the sake of example, we will consider two generic tests: `1a-0-basic` and `1a-1-hidden`.

Local Execution - Hidden Tests

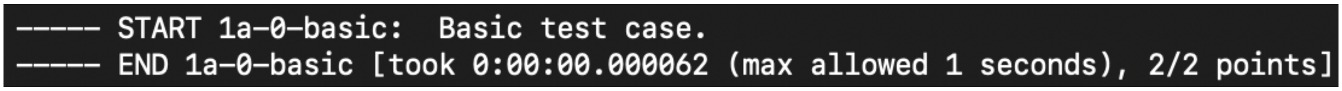
All hidden tests rely on files that are not provided to students. Therefore, the tests can only be run remotely. When a hidden test like `1a-1-hidden` is executed locally, it will produce the following result:



```
----- START 1a-1-hidden:  Test multiple instances of the same word in a sentence.
----- END 1a-1-hidden [took 0:00:00.011989 (max allowed 1 seconds), ???/3 points] (hidden test ungraded)
```

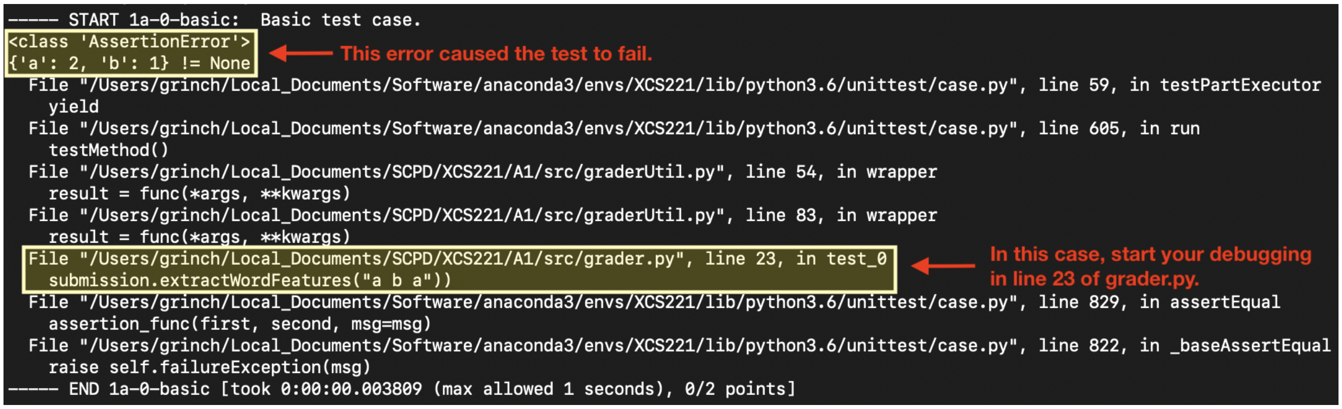
Local Execution - Basic Tests

When a basic test like `1a-0-basic` passes locally, the autograder will indicate success:



```
----- START 1a-0-basic:  Basic test case.
----- END 1a-0-basic [took 0:00:00.000062 (max allowed 1 seconds), 2/2 points]
```

When a basic test like `1a-0-basic` fails locally, the error is printed to the terminal, along with a stack trace indicating where the error occurred:



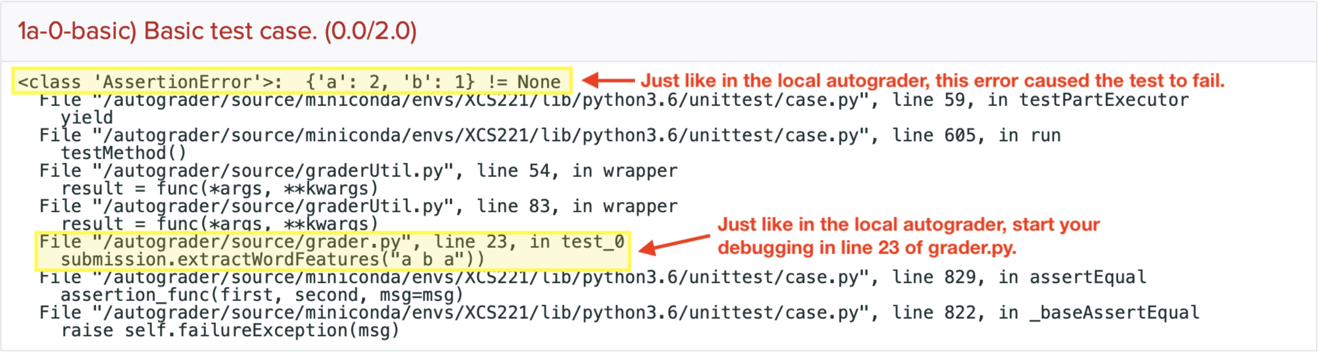
```
----- START 1a-0-basic:  Basic test case.
<class 'AssertionError'>
{'a': 2, 'b': 1} != None
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
----- END 1a-0-basic [took 0:00:00.003809 (max allowed 1 seconds), 0/2 points]
```

This error caused the test to fail.

In this case, start your debugging in line 23 of grader.py.

Remote Execution

Basic and hidden tests are treated the same by the remote autograder. Here are screenshots of failed basic and hidden tests. Notice that the same information (error and stack trace) is provided as the in local autograder, now for both basic and hidden tests.

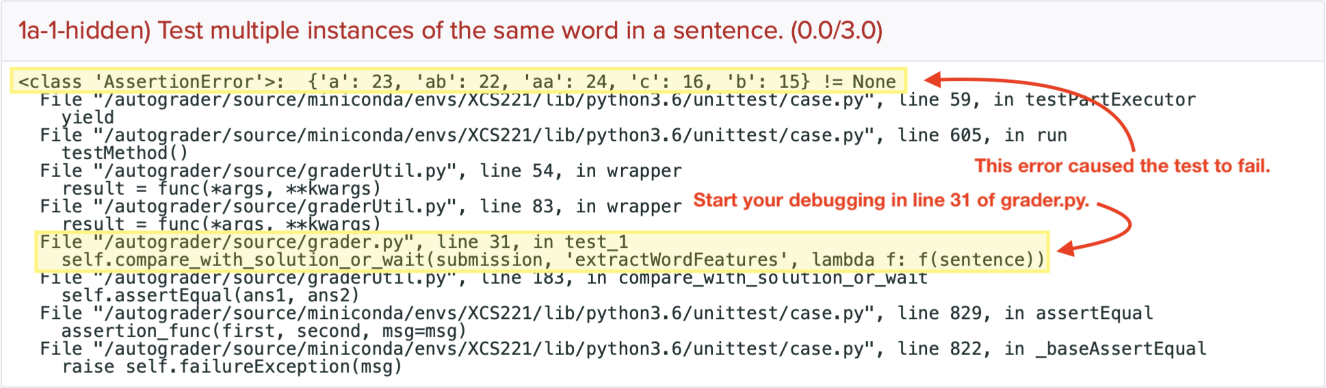


1a-0-basic) Basic test case. (0.0/2.0)

```
<class 'AssertionError'>:  {'a': 2, 'b': 1} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

Just like in the local autograder, this error caused the test to fail.

Just like in the local autograder, start your debugging in line 23 of grader.py.



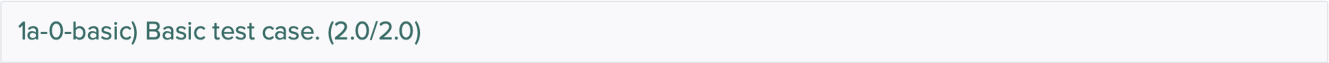
1a-1-hidden) Test multiple instances of the same word in a sentence. (0.0/3.0)

```
<class 'AssertionError'>:  {'a': 23, 'ab': 22, 'aa': 24, 'c': 16, 'b': 15} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 31, in test_1
    self.compare_with_solution_or_wait(submission, 'extractWordFeatures', lambda f: f(sentence))
  File "/autograder/source/graderUtil.py", line 183, in compare_with_solution_or_wait
    self.assertEqual(ans1, ans2)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

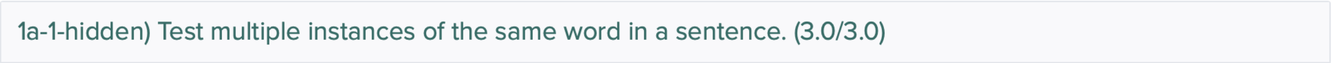
This error caused the test to fail.

Start your debugging in line 31 of grader.py.

Finally, here is what it looks like when basic and hidden tests pass in the remote autograder.



1a-0-basic) Basic test case. (2.0/2.0)



1a-1-hidden) Test multiple instances of the same word in a sentence. (3.0/3.0)

1. **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 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-linear/ds1_train,valid.csv`
- `src-linear/ds2_train,valid.csv`
- `src-linear/main.py`
- `src-linear/submission/__init__.py`
- `src-linear/submission/logreg.py`
- `src-linear/submission/gda.py`

Each file contains n examples, one example $(x^{(i)}, y^{(i)})$ per row. In particular, the i-th row contains columns $x_0^{(i)} \in \mathbb{R}$, $x_1^{(i)} \in \mathbb{R}$, 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.

Code Deliverables

- `src-linear/submission/logreg.py`
- `src-linear/submission/gda.py`
- `src-linear/submission/__init__.py`

(a) **[2 points (Written)]**

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) **[3 points (Coding)]** Follow the instructions in `src-linear/submission/logreg.py` to train a logistic regression classifier using Newton's Method. You will complete the `fit` and `predict` functions of the `LogisticRegression` class.

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.

To verify a correct implementation, run the autograder test case `1b-4-basic` to create a plot of the **validation data** with x_1 on the horizontal axis and x_2 on the vertical axis. This plot uses a different symbol for examples $x^{(i)}$ with $y^{(i)} = 0$ than for those with $y^{(i)} = 1$. On the same figure, it will also plot the decision boundary found by logistic regression (i.e, line corresponding to $p(y|x) = 0.5$).

The output plot should look similar to the following (no plot submission is required):

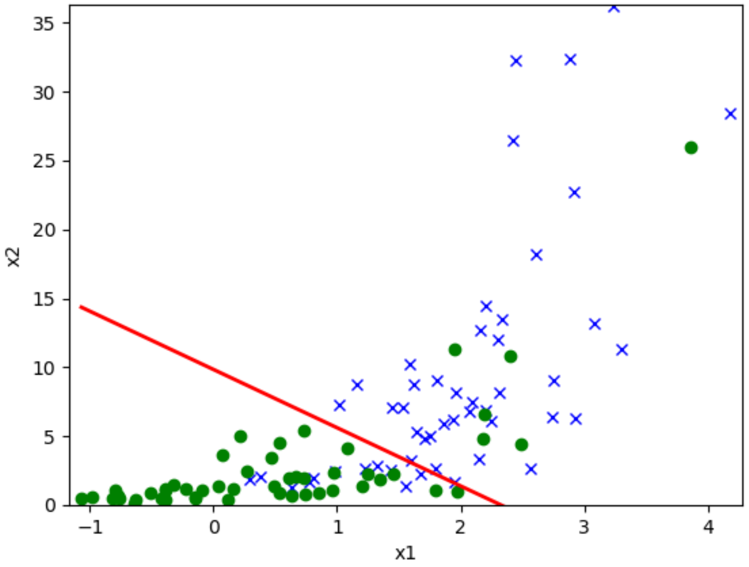


Figure 1: Separating hyperplane for logistic regression on validation set of Dataset 1 (Note: This is for reference only. You are not required to submit a plot.)

(c) **[2 points (Written)]** Recall that in GDA we model the joint distribution of (x, y) by the following equations:

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

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

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

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) **[3 points (Coding)]** In `src-linear/submission/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. More specifically, you complete the `fit` and `predict` functions for the `GDA` class.

Make sure to write your model's predictions on the validation set to the file specified in the code.

To verify a correct implementation, run the autograder test case `1e-4-basic` to create 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 $p(y|x) = 0.5$).

The output plot should look similar to the following (no plot submission is required):

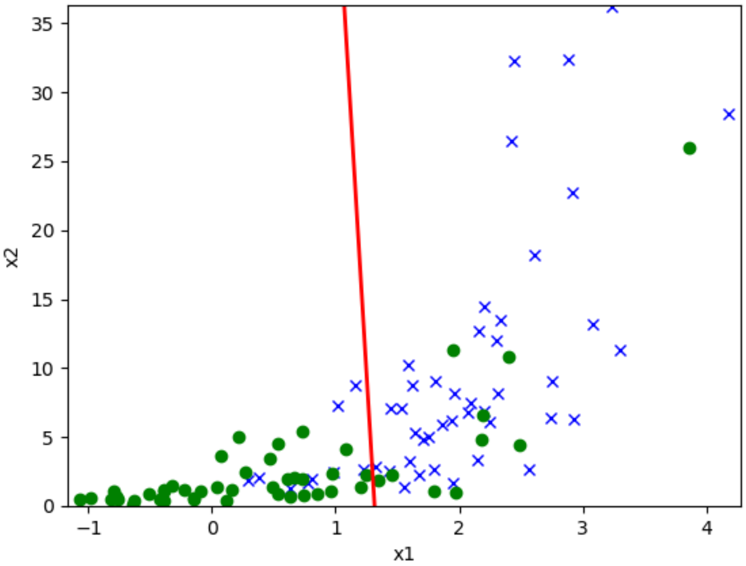


Figure 2: Separating hyperplane for GDA on the validation set for Dataset 1 (Note: This is for reference only. You are not required to submit a plot.)

(f) **[0.50 points (Written)]** 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. No plot submission is required.

(g) **[0.50 points (Written)]** Use autograder test case `1g-0-basic` to create GDA and logistic regression plots for Dataset 2. Compare the plots for Dataset 1 (from parts (b) and (e)) with the plots for Dataset 2. On which dataset does GDA seem to perform worse than logistic regression? Why might this be the case?

(h) **[0.50 points (Written)]** 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?

Once you are done with src-linear, please copy over your src-linear/submission/logreg.py to src-incomplete and src-imbalanced by running

(a) `cp ./src-linear/submission/logreg.py ./src-incomplete`

(b) `cp ./src-linear/submission/logreg.py ./src-imbalanced`

2. **Incomplete, Positive-Only Labels**

In this problem we will consider training binary classifiers in situations where we do not have full access to the labels. In particular, we consider a scenario, which is not too infrequent in real life, where we have labels only for a subset of the positive examples. All the negative examples and the rest of the positive examples are unlabelled.

We formalize the scenario as follows. Let $\{(x^{(i)}, t^{(i)})\}_{i=1}^n$ be a standard dataset of i.i.d distributed examples. Here $x^{(i)}$'s are the inputs/features and $t^{(i)}$ are the labels. Now consider the situation where $t^{(i)}$'s are not observed by us. Instead, we only observe the labels of some of the positive examples. Concretely, we assume that we observe $y^{(i)}$'s that are generated by

$$\begin{aligned}
\forall x, \quad & p(y^{(i)} = 1 \mid t^{(i)} = 1, x^{(i)} = x) = \alpha, \\
\forall x, \quad & p(y^{(i)} = 0 \mid t^{(i)} = 1, x^{(i)} = x) = 1 - \alpha \\
\forall x, \quad & p(y^{(i)} = 1 \mid t^{(i)} = 0, x^{(i)} = x) = 0, \\
\forall x, \quad & p(y^{(i)} = 0 \mid t^{(i)} = 0, x^{(i)} = x) = 1
\end{aligned}$$

where $\alpha \in (0, 1)$ is some unknown scalar. In other words, if the unobserved "true" label $t^{(i)}$ is 1, then with α chance we observe a label $y^{(i)} = 1$. On the other hand, if the unobserved "true" label $t^{(i)} = 0$, then we always observe the label $y^{(i)} = 0$.

Our final goal in the problem is to construct a binary classifier h of the true label t, with only access to the partial label y. In other words, we want to construct h such that $h(x^{(i)}) \approx p(t^{(i)} = 1 \mid x^{(i)})$ as closely as possible, using only x and y.

Real world example: Suppose we maintain a database of proteins which are involved in transmitting signals across membranes. Every example added to the database is involved in a signaling process, but there are many proteins involved in cross-membrane signaling which are missing from the database. It would be useful to train a classifier to identify proteins that should be added to the database. In our notation, each example $x^{(i)}$ corresponds to a protein, $y^{(i)} = 1$ if the protein is in the database and 0 otherwise, and $t^{(i)} = 1$ if the protein is involved in a cross-membrane signaling process and thus should be added to the database, and 0 otherwise.

For the rest of the question, we will use the dataset and starter code provided in the following files:

- `src-incomplete/{train,valid,test}.csv`
- `src-incomplete/submission.py`

Each file contains the following columns: x_1, x_2, y, and t. As in Problem 1, there is one example per row. The $y^{(i)}$'s are generated from the process defined above with some unknown α. **Note: We will be using the logistic regression classifier defined in problem 1 to apply binary classification for problem 2. If you haven't done so already, finish implementing the Logistic Regression class in src-linear/submission/logreg.py.**

Code Deliverables

- `src-incomplete/submission.py`
- `src-incomplete/logreg.py` ****Copied over from src-linear/submission/logreg.py**

(a) **[1 point (Coding)] Coding problem: ideal (fully observed) case**

First we will consider the hypothetical (and uninteresting) case, where we have access to the true t-labels for training. In `src-incomplete/submission.py`, write a logistic regression classifier that uses x_1 and x_2 as input features, and train it using the t-labels. More specifically you will implement the `fully_observed_predictions` function. We will ignore the y-labels for this part. Output the trained model's predictions on the **test set** to the file specified in the code.

The output plot should look similar to the following (no plot submission is required):

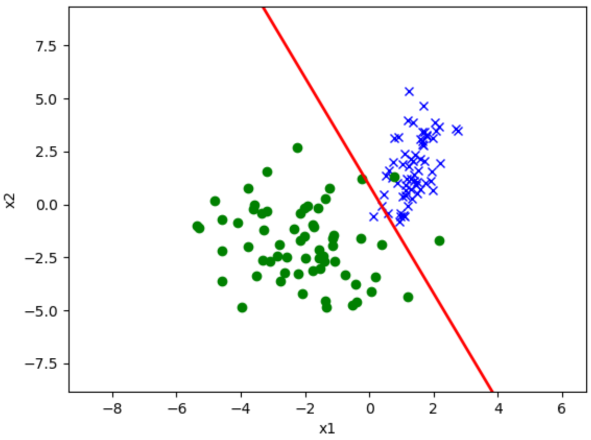


Figure 3: Separating hyperplane for logistic regression on training set using fully observed predictions (Note: This is for reference only. You are not required to submit a plot.)

(b) **[1 point (Coding)] Coding problem: The naive method on partial labels**

We now consider the case where the t-labels are unavailable, so you only have access to the y-labels at training time. Extend your code in `src-incomplete/submission.py` to re-train the classifier (still using x_1 and x_2 as input features), but using the y-labels only. More specifically, you will implement the `naive_partial_labels_predictions` function. Output the predictions on the **test set** to the appropriate file (as described in the code comments).

Note that the algorithm should learn a function $h(\cdot)$ that approximately predicts the probability $p(y^{(i)} = 1 \mid x^{(i)})$. Also note that we expect it to perform poorly on predicting the probability of interest, namely $p(t^{(i)} = 1 \mid x^{(i)})$.

The output plot should look similar to the following (no plot submission is required):

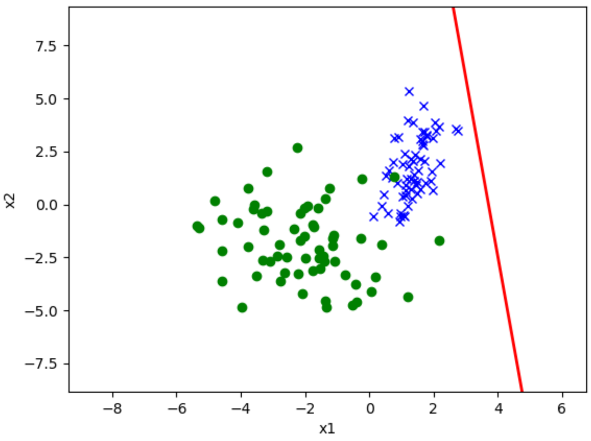


Figure 4: Separating hyperplane for logistic regression on training set using naive partially observed labels (Note: This is for reference only. You are not required to submit a plot.)

In the following sub-questions we will attempt to solve the problem with only partial observations. That is, we only have access to $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, and will try to predict $p(t^{(i)} = 1 \mid x^{(i)})$.

(c) **[1 point (Written)] Warm-up with Bayes rule**

Show that under our assumptions, for any i,

$$p(t^{(i)} = 1 \mid y^{(i)} = 1, x^{(i)}) = 1 \tag{1}$$

That is, observing a positive partial label $y^{(i)} = 1$ tells us for sure the hidden true label is 1. Use Bayes rule to derive this (an informal explanation will not earn credit).

(d) **[1.50 points (Written)]** Show that for any example, the probability that true label $t^{(i)}$ is positive is $1/\alpha$ times the probability that the partial label is positive. That is, show that

$$p(t^{(i)} = 1 \mid x^{(i)}) = \frac{1}{\alpha} \cdot p(y^{(i)} = 1 \mid x^{(i)}) \tag{2}$$

Note that the equation above suggests that if we know the value of α, then we can convert a function $h(\cdot)$ that approximately predicts the probability $h(x^{(i)}) \approx p(y^{(i)} = 1 \mid x^{(i)})$ into a function that approximately predicts $p(t^{(i)} = 1 \mid x^{(i)})$ by multiplying the factor $1/\alpha$.

(e) **[2 points (Written)] Estimating α**

The solution to estimate $p(t^{(i)} \mid x^{(i)})$ outlined in the previous sub-question requires the knowledge of α which we don't have. Now we will design a way to estimate α based on the function $h(\cdot)$ that approximately predicts $p(y^{(i)} = 1 \mid x^{(i)})$ (which we obtained in part b).

To simplify the analysis, let's assume that we have magically obtained a function $h(x)$ that perfectly predicts the value of $p(y^{(i)} = 1 \mid x^{(i)})$, that is, $h(x^{(i)}) = p(y^{(i)} = 1 \mid x^{(i)})$.

We make the crucial assumption that $p(t^{(i)} = 1 \mid x^{(i)}) \in \{0, 1\}$. This assumption means that the process of generating the "true" label $t^{(i)}$ is a noise-free process. This assumption is not very unreasonable to make. Note, we are NOT assuming that the observed label $y^{(i)}$ is noise-free, which would be an unreasonable assumption!

Now we will show that:

$$\alpha = \mathbb{E}[h(x^{(i)}) \mid y^{(i)} = 1] \tag{3}$$

To show this, prove that $h(x^{(i)}) = \alpha$ when $y^{(i)} = 1$, and $h(x^{(i)}) = 0$ when $y^{(i)} = 0$.

The above result motivates the following algorithm to estimate α by estimating the RHS of the equation above using samples: Let V_+ be the set of labeled (and hence positive) examples in the validation set V, given by $V_+ = \{x^{(i)} \in V \mid y^{(i)} = 1\}$.

Then we use

$$\alpha \approx \frac{1}{\mid V_+ \mid} \sum_{x^{(i)} \in V_+} h(x^{(i)}).$$

to estimate α. (You will be asked to implement this algorithm in the next sub-question. For this sub-question, you only need to show equation (3). Moreover, this sub-question may be slightly harder than other sub-questions.)

(f) **[2 points (Coding)] Coding problem.**

Using the validation set, estimate the constant α by averaging your classifier's predictions over all labeled examples in the validation set:[1]

$$\alpha \approx \frac{1}{\mid V_+ \mid} \sum_{x^{(i)} \in V_+} h(x^{(i)}).$$

Add code in `src-incomplete/submission.py` to rescale your predictions $h(y^{(i)} = 1 \mid x^{(i)})$ of the classifier that is obtained from part b, using the equation (2) obtained in part (d) and using the estimated value for α. More specifically implement the `find_alpha_and_plot_correction` function.

The output plot should look similar to the following (no plot submission is required):

[1] There is a reason to use the validation set, instead of the training set, to estimate the α. However, for the purpose of this question, we sweep the subtlety here under the rug, and you don't need to understand the difference between the two for this question.

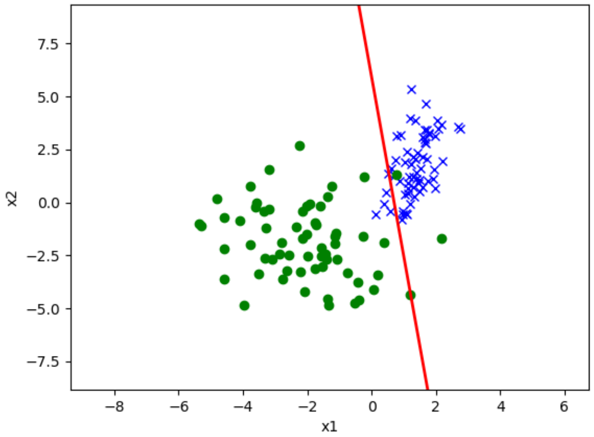


Figure 5: Separating hyperplane for logistic regression on training set using using corrected alpha value from validation set (Note: This is for reference only. You are not required to submit a plot.)

Remark: We saw that the true probability $p(t \mid x)$ was only a constant factor away from $p(y \mid x)$. This means, if our task is to only rank examples (*i.e.* sort them) in a particular order (e.g, sort the proteins in order of being most likely to be involved in transmitting signals across membranes), then in fact we do not even need to estimate α. The rank based on $p(y \mid x)$ will agree with the rank based on $p(t \mid x)$.

3. Learning Imbalanced Dataset

In this problem, we study how to learn a classifier from an imbalanced dataset, where the marginal distribution of the classes/labels are imbalanced. Imbalanced datasets are ubiquitous in real-world applications. For example, in the spam detection problem, the training dataset usually has only a small fraction of spam emails (positive examples) but a large fraction of ordinary emails (negative examples). For simplicity, we consider binary classification problem where the labels are in $\{0, 1\}$ and the number of positive examples is much smaller than the number of negative examples.

Note: We will be using the logistic regression classifier defined in problem 1. If you haven't done so already, finish implementing the Logistic Regression class in src-linear/submission/logreg.py

Code Deliverables

- `src-imbalanced/submission.py`
- `src-imbalanced/logreg.py` ****Copied over from src-linear/submission/logreg.py**

(a) Evaluation metrics

In this sub-question, we discuss the common evaluation metrics for imbalanced dataset. Suppose we have a validation dataset and for some $\rho \in (0, 1)$, we assume that ρ fraction of the validation examples are positive examples (with label 1), and $1 - \rho$ fraction of them are negative examples (with label 0).

Define the accuracy as

$$A \triangleq \frac{\#\text{examples that are predicted correctly by the classifier}}{\#\text{examples}}$$

(i) **[1 point (Written)]** Show that for any dataset with ρ fraction of positive examples and $1 - \rho$ fraction of negative examples, there exists a (trivial) classifier with accuracy at least $1 - \rho$.

The statement above suggests that the accuracy is not an ideal evaluation metric when ρ is close to 0. E.g., imagine that for spam detection ρ can be smaller than 1%. The statement suggests there is a trivial classifier that gets more than 99% accuracy. This could be misleading — 99% seems to be almost perfect, but actually you don't need to learn anything from the dataset to achieve it.

Therefore, for imbalanced dataset, we need more informative evaluation metrics. We define the number of true positive, true negative, false positive, false negative examples as

$$\begin{aligned}
TP &\triangleq \#\text{ positive examples with a correct (positive) prediction}\\
TN &\triangleq \#\text{ negative examples with a correct (negative) prediction}\\
FP &\triangleq \#\text{ negative examples with a incorrect (positive) prediction}\\
FN &\triangleq \#\text{ positive examples with a incorrect (negative) prediction}
\end{aligned}$$

Define the accuracy of positive examples as

$$A_1 \triangleq \frac{TP}{TP + FN} = \frac{\#\text{ positive examples with a correct (positive) prediction}}{\#\text{ positive examples}}$$

Define the accuracy of negative examples as

$$A_0 \triangleq \frac{TN}{TN + FP} = \frac{\#\text{ negative examples with a correct (negative) prediction}}{\#\text{ negative examples}}$$

We define the balanced accuracy as

$$\overline{A} \triangleq \frac{1}{2}\left(A_0 + A_1\right) \tag{4}$$

With these notations, we can verify that the accuracy is equal to $A = \frac{TP+TN}{TP+TN+FP+FN}$.

(ii) **[2 points (Written)]** Show that

$$\rho = \frac{TP + FN}{TP + TN + FP + FN}$$

and

$$A = \rho \cdot A_1 + (1 - \rho)A_0 \tag{5}$$

Comparing equation (4) and (5), we can see that the accuracy and balanced accuracy are both linear combination of A_0 and A_1 but with different weighting.

(iii) **[2 points (Written)]** Show that the trivial classifier you constructed for part (i) has balanced accuracy 50%.

Partly because of (iii), the balanced accuracy $\overline{A}$ is often a preferable evaluation metric than the accuracy A. Sometimes people also report the accuracies for the two classes (A_0 and A_1) to demonstrate the performance for each class.

(b) **[3 points (Coding)] Coding problem: vanilla logistic regression**

First, we use the vanilla logistic regression to learn an imbalanced dataset. For the rest of the question, we will use the dataset and starter code provided in the following files:

- `src-imbalanced/train,validation.csv`
- `src-imbalanced/submission.py`

Please complete `apply_logistic_regression` and `calculate_accuracies` functions.

Each file contains n examples, one example $(x^{(i)}, y^{(i)})$ per row. x is two-dimensional, i.e., the i-th row contains columns $x_1^{(i)} \in \mathbb{R}$, $x_2^{(i)} \in \mathbb{R}$, and $y^{(i)} \in \{0, 1\}$. Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ be our training dataset. $\mathcal{D}$ has ρn examples with label 1 and $(1 - \rho)n$ with label 0. In the dataset we constructed, $\rho = 1/11$.

You will train a linear classifier $h_\theta(x)$ with average empirical loss for logistical regression, where $h_\theta(x) = g(\theta^T x), g(z) = 1/(1 + e^{-z})$, similar to Problem 1 Part a:

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

After obtaining the classifier, compute the classifier's accuracy (A), balanced accuracy ($\overline{A}$), accuracies for the two classes (A_0, A_1) on the validation dataset, and report them in the writeup. You are expected to observe that the minority class (positive class) has significantly lower accuracy than the majority class.

The output plot should look similar to the following (no plot submission is required):

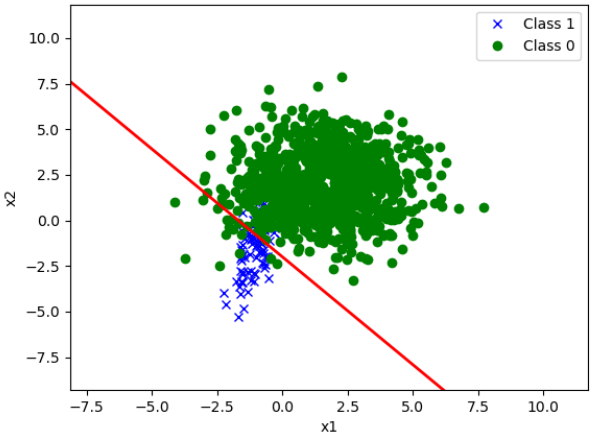


Figure 6: Validation set with x_1 on the horizontal axis and x_2 on the vertical axis. The decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5).

(c) **[2 points (Written)] Re-sampling/Re-weighting Logistic Regression**

The relatively low accuracy for the minority class and the resulting low balanced accuracy are undesirable for many applications. Various methods have been proposed to improve the accuracy for the minority class, and learning imbalanced datasets is an active open research direction. Here we introduce a simple and classical re-sampling/re-weighting technique that helps improve the balanced accuracy in most of the cases.

We observe that the logistic regression algorithm works well for the accuracy but not for the balanced accuracy. Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ denote the existing training dataset. We will create a new dataset $\mathcal{D}'$ such that the accuracy on $\mathcal{D}'$ is the same as the balanced accuracy on $\mathcal{D}$.

Assume $\rho < 1/2$ without loss of generality, and let $\kappa = \frac{\rho}{1-\rho}$. This means the number of positive examples is κ times the number of negative examples in $\mathcal{D}$. Assume for convenience $1/\kappa$ is an integer.[2] Let $\mathcal{D}'$ be the dataset that contain each negative example once and $1/\kappa$ repetitions of each positive example in $\mathcal{D}$. Then we will apply the logistic regression on the new dataset $\mathcal{D}'$.

Prove that for any classifier, the balanced accuracy on $\mathcal{D}$ is equal to the accuracy on $\mathcal{D}'$. Moreover, **show that** the average empirical loss for logistical regression on the dataset $\mathcal{D}'$ is equal to

$$J(\theta) = -\frac{1+\kappa}{2n} \sum_{i=1}^{n} w^{(i)} \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right),$$

where $w^{(i)} = 1$ if $y^{(i)} = 0$ and $w^{(i)} = 1/\kappa$ if $y^{(i)} = 1$.

Observe effectively we are re-weighting the loss function for each example by some constant that depends on the frequency of the class it belongs to.

(d) **[3 points (Coding)] Coding problem: re-weighting minority class**

In `src-imbalanced/submission.py`, implement the logistic regression algorithm on $\mathcal{D}'$. Compute and report the classifier's accuracy (A), balanced accuracy ($\overline{A}$), accuracies for the two classes (A_0, A_1) on the validation dataset. You are expected to see that the accuracy of minority class (class 1) improved significantly whereas that of the majority class dropped compared to vanilla logistic regression. However, the balanced accuracy is significantly greater than that of the vanilla logistic regression.

Please adjust the `apply_logistic_regression` you worked with in 3b to capture the reweighted minority class solution. Then fill in the `upsample_minority_class` function.

The output plot should look similar to the following (no plot submission is required):

[2] otherwise we can round up to the nearest integer with introducing a slight approximation.

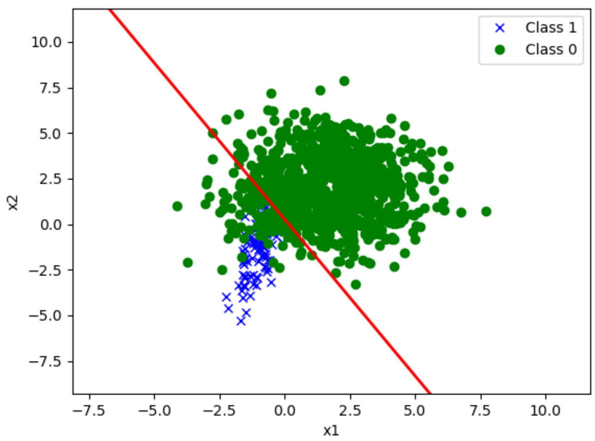


Figure 7: Validation set with x_1 on the horizontal axis and x_2 on the vertical axis. The decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5).

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the `README.md` for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

1.a

Since $g'(z) = g(z)(1 - g(z))$ and $h(x) = g(\theta^T x)$, it follows that $\partial h(x)/\partial\theta_k = h(x)(1 - h(x))x_k$.

Letting $h_\theta(x^{(i)}) = g(\theta^T x^{(i)}) = 1/(1 + \exp(-\theta^T x^{(i)}))$, we have

$$\frac{\partial \log h_\theta(x^{(i)})}{\partial\theta_k} =$$

$$\frac{\partial \log(1 - h_\theta(x^{(i)}))}{\partial\theta_k} =$$

Substituting into our equation for $J(\theta)$, we have

$$\frac{\partial J(\theta)}{\partial\theta_k} =$$

Consequently, the (k, l) entry of the Hessian is given by

$$H_{kl} = \frac{\partial^2 J(\theta)}{\partial\theta_k \partial\theta_l} =$$

Using the fact that $X_{ij} = x_i x_j$ if and only if $X = xx^T$, we have

$$H =$$

To prove that H is positive semi-definite, show $z^T H z \geq 0$ for all $z \in \mathbb{R}^d$.

$$z^T H z =$$

1.c

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:

$$\begin{aligned}
p(y = 1|x; \mathcal{H}) &= \frac{p(x|y = 1; \mathcal{H})p(y = 1; \mathcal{H})}{p(x; \mathcal{H})} \\
&= \frac{p(x|y = 1; \mathcal{H})p(y = 1; \mathcal{H})}{p(x|y = 1; \mathcal{H})p(y = 1; \mathcal{H}) + p(x|y = 0; \mathcal{H})p(y = 0; \mathcal{H})} \\
&=
\end{aligned}$$

1.d

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

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

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

For ϕ:

$$\frac{\partial\ell}{\partial\phi}=$$

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

For μ_0:

Hint: Remember that Σ (and thus Σ^{-1}) is symmetric.

$$\nabla_{\mu_0}\ell=$$

Setting this gradient to zero gives the maximum likelihood estimate for μ_0.

For μ_1:

Hint: Remember that Σ (and thus Σ^{-1}) is symmetric.

$$\nabla_{\mu_1}\ell=$$

Setting this gradient to zero gives the maximum likelihood estimate for μ_1.

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 $S_n^{-1}=\Sigma_n$.

Hint: You may need the following identities:

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

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

$$\nabla_S\ell=$$

Next, substitute $\Sigma=S^{-1}$. Setting this gradient to zero gives the required maximum likelihood estimate for Σ.

1.f

1.g

1.h

2.c

2.d

2.e

3.a

3.c