CS 229, Fall 2025
Problem Set #2

YOUR NAME HERE (`YOUR SUNET HERE`)

Due Wednesday, October 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/87361/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 22 at 11:59 pm. If you submit after Wednesday, October 22 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. **[20 points] Bayesian Interpretation of Regularization**

Background: In Bayesian statistics, almost every quantity is a random variable, which can either be observed or unobserved. For instance, parameters θ are generally unobserved random variables, and data x and y are observed random variables. The joint distribution of all the random variables is also called the *model* (*e.g.,* $p(x, y, \theta)$). Every unknown quantity can be estimated by conditioning the model on all the observed quantities. Such a conditional distribution over the unobserved random variables, conditioned on the observed random variables, is called the *posterior distribution*. For instance $p(\theta|x, y)$ is the posterior distribution in the machine learning context. A consequence of this approach is that we are required to endow our model parameters, *i.e.,* $p(\theta)$, with a *prior distribution*. The prior probabilities are to be assigned *before* we see the data—they capture our prior beliefs of what the model parameters might be before observing any evidence.

In the purest Bayesian interpretation, we are required to keep the entire posterior distribution over the parameters all the way until prediction, to come up with the *posterior predictive distribution*, and the final prediction will be the expected value of the posterior predictive distribution. However in most situations, this is computationally very expensive, and we settle for a compromise that is *less pure* (in the Bayesian sense).

The compromise is to estimate a point value of the parameters (instead of the full distribution) which is the mode of the posterior distribution. Estimating the mode of the posterior distribution is also called *maximum a posteriori estimation* (MAP). That is,

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta|x, y).$$

Compare this to the *maximum likelihood estimation* (MLE) we have seen previously:

$$\theta_{\text{MLE}} = \arg\max_{\theta} p(y|x, \theta).$$

In this problem, we explore the connection between MAP estimation, and common regularization techniques that are applied with MLE estimation. In particular, you will show how the choice of prior distribution over θ (*e.g.,* Gaussian or Laplace prior) is equivalent to different kinds of regularization (*e.g.,* L_2, or L_1 regularization). You will also explore how regularization strengths affect generalization in part (d).

(a) [3 points] Show that $\theta_{\text{MAP}} = \text{argmax}_{\theta}\, p(y|x, \theta)p(\theta)$ if we assume that $p(\theta) = p(\theta|x)$. The assumption that $p(\theta) = p(\theta|x)$ will be valid for models such as linear regression where the input x are not explicitly modeled by θ. (Note that this means x and θ are marginally independent, but not conditionally independent when y is given.)

Answer:

$$\theta_{\mathsf{MAP}} = \arg\max_{\theta}\, p(\theta \mid x, y)$$

Bayes:

$$p(\theta \mid x, y) = \frac{p(y \mid x, \theta)\, p(\theta \mid x)}{p(y \mid x)}$$

$p(y \mid x)$ can be dropped as it does not depend on θ:

$$\boxed{\theta_{\mathsf{MAP}} = \arg\max_{\theta}\, p(y \mid x, \theta)\, p(\theta)}$$

(b) [5 points] Recall that L_2 regularization penalizes the L_2 norm of the parameters while minimizing the loss (*i.e.,* negative log likelihood in case of probabilistic models). Now we will show that MAP estimation with a zero-mean Gaussian prior over θ, specifically $\theta \sim \mathcal{N}(0, \eta^2 I)$, is equivalent to applying L_2 regularization with MLE estimation. Specifically, show that for some scalar λ,

$$\theta_{\text{MAP}} = \arg\min_\theta - \log p(y|x, \theta) + \lambda ||\theta||_2^2. \tag{1}$$

Also, what is the value of λ?

Answer:

$$\theta_{MAP} = \arg\max_\theta p(y \mid x, \theta) p(\theta)$$

Maximizing the product is the same as minimizing the negative log-likelihood

$$\theta_{MAP} = \arg\max_\theta p(y \mid x, \theta) p(\theta) = \arg\min_\theta (-\log(p(y \mid x, \theta) p(\theta)))$$

$$= \arg\min_\theta (-\log p(y \mid x, \theta) - \log p(\theta))$$

$$\log p(y \mid x, \theta) = \log \left(\frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{(y - x^T\theta)^2}{2\sigma^2} \right) \right) = -\frac{(y - x^T\theta)^2}{2\sigma^2} + \text{constant}$$

With $\theta \sim \mathcal{N}(0, \eta^2 I)$, log of prior:

$$\log p(\theta) = \log \left(\frac{1}{(2\pi\eta^2)^{d/2}} \exp\left(-\frac{\theta^T\theta}{2\eta^2} \right) \right) = -\frac{\|\theta\|_2^2}{2\eta^2} + \text{constant}$$

$$\arg\min_\theta (-\log p(y \mid x, \theta) - \log p(\theta)) = \arg\min_\theta \frac{(y - x^T\theta)^2}{2\sigma^2} + \frac{\|\theta\|_2^2}{2\eta^2} + \text{constant}$$

$$\theta_{MAP} = \arg\min_\theta \frac{(y - x^T\theta)^2}{2\sigma^2} + \frac{\|\theta\|_2^2}{2\eta^2}$$

$$\boxed{\theta_{MAP} = \arg\min_\theta \left((y - x^T\theta)^2 + \lambda \|\theta\|_2^2 \right), \qquad \lambda = \frac{2\sigma^2}{2\eta^2} = \frac{\sigma^2}{\eta^2}}$$

(c) [7 points] Now consider a specific instance, a linear regression model given by $y = \theta^T x + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Assume that the random noise $\epsilon^{(i)}$ is independent for every training example $x^{(i)}$. Like before, assume a Gaussian prior on this model such that $\theta \sim \mathcal{N}(0, \eta^2 I)$. For notation, let X be the design matrix of all the training example inputs where each row vector is one example input, and $\vec{y}$ be the column vector of all the example outputs.

Come up with a closed form expression for θ_{MAP}.

[**Hint:** Use what we proved in part (b).]

Answer:

Let J_λ be the regularized loss:

$$J_\lambda(\theta) = J(\theta) + \lambda R(\theta)$$

with $\lambda \geq 0$ as a regularization parameter, and $R(\theta)$ as the regularizer. For linear regression and ℓ_2 norm

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

Gradients

$$\nabla_\theta\left(\tfrac{1}{2}\|X\theta - y\|_2^2\right) = X^\top(X\theta - y)$$

$$\nabla_\theta\left(\tfrac{\lambda}{2}\|\theta\|_2^2\right) = \lambda\,\theta$$

$$\nabla_\theta J_\lambda(\theta) = X^\top(X\theta - y) + \lambda\,\theta$$

Set $\nabla_\theta J_\lambda(\theta) = 0$

$$0 = X^\top(X\theta - y) + \lambda\,\theta \qquad \therefore \qquad X^T y = \lambda\theta + X^T X\theta$$

$$\boxed{\theta = (X^T X + \lambda I)}$$

(d) [5 points] Next, consider the Laplace distribution, whose density is given by

$$f_{\mathcal{L}}(z|\mu, b) = \frac{1}{2b}\exp\left(-\frac{|z-\mu|}{b}\right).$$

As before, consider a linear regression model given by $y = x^T\theta + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Assume a Laplace prior on this model, where each parameter θ_i is marginally independent, and is distributed as $\theta_i \sim \mathcal{L}(0, b)$.

Show that θ_{MAP} in this case is equivalent to the solution of linear regression with L_1 regularization, whose loss is specified as

$$J(\theta) = ||X\theta - \vec{y}||_2^2 + \gamma||\theta||_1$$

Also, what is the value of γ?

Note: A closed form solution for linear regression problem with L_1 regularization does not exist. To optimize this, we use gradient descent with a random initialization and solve it numerically.

Answer:

For each parameter θ_i

$$p(\theta_i) = \frac{1}{2b}\exp\left(-\frac{|\theta_i|}{b}\right)$$

As each parameter is marginally independent, the probability density function is

$$p(\theta) = \prod_i^d p(\theta_i) = \left(\frac{1}{2b}\right)^d \exp\left(-\frac{1}{b}\sum_i^d |\theta_i|\right)$$

and the L1 norm is $\sum_i^d |\theta_i| = \|\theta\|_1$, so

$$p(\theta) = \left(\frac{1}{2b}\right)^d \exp\left(-\frac{\|\theta\|_1}{b}\right)$$

Similar to previous exercise, maximizing the product is the same as minimizing the negative log-likelihood

$$\theta_{\mathsf{MAP}} = \arg\max_\theta p(y \mid X, \theta)p(\theta) = \arg\min_\theta\left(-\log(p(y \mid X, \theta)p(\theta))\right)$$

$$= \arg\min_\theta \left(-\log p(y \mid X, \theta) - \log p(\theta)\right)$$

$$\log p(y \mid X, \theta) = \log \left(\frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{\|(y - X\theta)\|_2^2}{2\sigma^2} \right) \right) = -\frac{\|(y - X\theta)\|_2^2}{2\sigma^2} + \textsf{constant}$$

$$\log p(\theta) = \log \left((\frac{1}{2b})^d \exp\left(-\frac{\|\theta\|_1}{b} \right) \right) = -\frac{\|\theta\|_1}{b} + \textsf{constant}$$

$$\arg\min_\theta \left(-\log p(y \mid X, \theta) - \log p(\theta)\right) = \arg\min_\theta \frac{\|(y - X\theta)\|_2^2}{2\sigma^2} + \frac{\|\theta\|_1}{b} + \textsf{constant}$$

$$\theta_{\textsf{MAP}} = \arg\min_\theta \frac{\|(y - X\theta)\|_2^2}{2\sigma^2} + \frac{\|\theta\|_1}{b}$$

$$\boxed{\theta_{\textsf{MAP}} = \arg\min_\theta \|(y - X\theta)\|_2^2 + \gamma\|\theta\|_1, \qquad \gamma = 2\frac{\sigma^2}{b}}$$

Remark: Linear regression with L_2 regularization is also commonly called *Ridge regression*, and when L_1 regularization is employed, is commonly called *Lasso regression*. These regularizations can be applied to any Generalized Linear models just as above (by replacing $\log p(y|x, \theta)$ with the appropriate family likelihood). Regularization techniques of the above type are also called *weight decay*, and *shrinkage*. The Gaussian and Laplace priors encourage the parameter values to be closer to their mean (*i.e.,* zero), which results in the shrinkage effect.

Remark: Lasso regression (*i.e.,* L_1 regularization) is known to result in sparse parameters, where most of the parameter values are zero, with only some of them non-zero.

2. **[25 points] Linear Classifiers (GDA)**

In PSET 1, you covered logistic regression in problem 3. In this problem, we apply a generative linear classifier, Gaussian discriminant analysis (GDA), on the same datasets. 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 the same 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/gda.py`

Recall that 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 $y^{(i)} \in \{0, 1\}$.

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

$$p(y) = \begin{cases} \phi & \text{if } y = 1 \\ 1 - \phi & \text{if } y = 0 \end{cases} \tag{2}$$

$$p(x|y=0) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right) \tag{3}$$

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

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:

$$p(y = 1 \mid x; \phi, \mu_0, \mu_1, \Sigma) = \frac{p(x \mid y=1)p(y=1)}{p(x \mid y=1)p(y=1) + p(x \mid y=0)p(y=0)}$$

Rewrite the posterior following $\frac{a}{a+b} = \frac{1}{a^{-1}}\frac{1}{a+b} = \frac{1}{1+a^{-1}b}$

$$p(y = 1 \mid x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \frac{p(x|y=0)p(y=0)}{p(x|y=1)p(y=1)}}$$

Then, to comply with the logistics form:

$$p(y = 1 \mid x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp\log\left(\frac{p(x|y=0)p(y=0)}{p(x|y=1)p(y=1)}\right)}$$

$$\log\frac{p(x\mid y=0)p(y=0)}{p(x\mid y=1)p(y=1)} = \log\frac{p(x\mid y=0)}{p(x\mid y=1)} + \log\frac{p(y=0)}{p(y=1)}$$

And

$$p(x\mid y=i) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu_i)^\top\Sigma^{-1}(x-\mu_i)\right), \quad i\in\{0,1\}.$$

So

$$\begin{aligned}\log\frac{p(x\mid y=0)}{p(x\mid y=1)} &= -\frac{1}{2}\Big[(x-\mu_0)^\top\Sigma^{-1}(x-\mu_0)-(x-\mu_1)^\top\Sigma^{-1}(x-\mu_1)\Big]\\ &= x^\top\Sigma^{-1}(\mu_0-\mu_1)+\tfrac{1}{2}(\mu_1^\top\Sigma^{-1}\mu_1-\mu_0^\top\Sigma^{-1}\mu_0)\\ \log\frac{p(y=0)}{p(y=1)} &= \log\frac{1-\phi}{\phi} = -\log\frac{\phi}{1-\phi}\end{aligned}$$

$$\exp\log\left(\frac{p(x\mid y=0)p(y=0)}{p(x\mid y=1)p(y=1)}\right) = \exp\left((\mu_0-\mu_1)^\top\Sigma^{-1}x+\tfrac{1}{2}(\mu_1^\top\Sigma^{-1}\mu_1-\mu_0^\top\Sigma^{-1}\mu_0)+\log\frac{1-\phi}{\phi}\right)$$

Then, the values of θ and θ_0 to match the logistic form are:

$$\boxed{\theta = \Sigma^{-1}(\mu_1-\mu_0)}$$

$$\boxed{\theta_0 = \tfrac{1}{2}(\mu_0^\top\Sigma^{-1}\mu_0-\mu_1^\top\Sigma^{-1}\mu_1)+\log\frac{\phi}{1-\phi}}$$

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

$$\phi = \frac{1}{n}\sum_{i=1}^n 1\{y^{(i)}=1\} \tag{4}$$

$$\mu_0 = \frac{\sum_{i=1}^n 1\{y^{(i)}=0\}x^{(i)}}{\sum_{i=1}^n 1\{y^{(i)}=0\}} \tag{5}$$

$$\mu_1 = \frac{\sum_{i=1}^n 1\{y^{(i)}=1\}x^{(i)}}{\sum_{i=1}^n 1\{y^{(i)}=1\}} \tag{6}$$

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

The log-likelihood of the data is

$$\begin{aligned}\ell(\phi,\mu_0,\mu_1,\Sigma) &= \log\prod_{i=1}^n p(x^{(i)},y^{(i)};\phi,\mu_0,\mu_1,\Sigma) \qquad (7)\\ &= \log\prod_{i=1}^n p(x^{(i)}|y^{(i)};\mu_0,\mu_1,\Sigma)p(y^{(i)};\phi).\end{aligned}$$

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.)

Answer:

$$
\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{n} \log p(x^{(i)}, y^{(i)}; \theta) \\
&= \sum_{i=1}^{n} \log p(x^{(i)} \mid y^{(i)}; \mu_0, \mu_1, \Sigma) + \sum_{i=1}^{n} \log p(y^{(i)}; \phi) \\
&= \sum_{i=1}^{n} \Big[-\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\big(x^{(i)} - \mu_{y^{(i)}}\big)^{\top}\Sigma^{-1}\big(x^{(i)} - \mu_{y^{(i)}}\big) \Big] \\
&\quad + \sum_{i=1}^{n} \Big[y^{(i)} \log\phi + \big(1 - y^{(i)}\big)\log(1-\phi) \Big]
\end{aligned}
$$

Constants can be ignored when maximizing the log-likelihood, so we have

$$
\begin{aligned}
\ell(\theta) = -\frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n} \big(x^{(i)} - \mu_{y^{(i)}}\big)^{\top}\Sigma^{-1}\big(x^{(i)} - \mu_{y^{(i)}}\big) \\
+ \sum_{i=1}^{n} \Big[y^{(i)} \log\phi + \big(1 - y^{(i)}\big)\log(1-\phi) \Big]
\end{aligned}
$$

For ϕ, we have

$$
\begin{aligned}
\frac{\partial}{\partial\phi}\ell(\theta) &= \sum_{i=1}^{n} \Big[\frac{y^{(i)}}{\phi} - \frac{1 - y^{(i)}}{1-\phi} \Big] \\
&= \frac{1}{\phi(1-\phi)} \Big[\sum_{i=1}^{n} y^{(i)} - n\phi \Big]
\end{aligned}
$$

Setting this to zero,

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

For μ_0, we care for those with $y^{(i)} = 0$, so

$$
\ell(\mu_0) = -\frac{1}{2}\sum_{i=1}^{n} \mathbf{1}\{y^{(i)} = 0\}\big(x^{(i)} - \mu_0\big)^{\top}\Sigma^{-1}\big(x^{(i)} - \mu_0\big) + \text{const}
$$

$$
\begin{aligned}
\frac{\partial\ell}{\partial\mu_0} &= -\frac{1}{2}\sum_{i=1}^{n} \mathbf{1}\{y^{(i)} = 0\} \cdot \big(-2\Sigma^{-1}(x^{(i)} - \mu_0)\big) \\
&= \Sigma^{-1}\sum_{i=1}^{n} \mathbf{1}\{y^{(i)} = 0\}\big(x^{(i)} - \mu_0\big)
\end{aligned}
$$

Setting this to zero,

$$
\sum_{i=1}^{n} \mathbf{1}\{y^{(i)} = 0\}x^{(i)} = \left(\sum_{i=1}^{n} \mathbf{1}\{y^{(i)} = 0\}\right)\mu_0 \;\therefore\; \boxed{\mu_0 = \frac{\sum_{i=1}^{n} \mathbf{1}\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^{n} \mathbf{1}\{y^{(i)} = 0\}}}
$$

Similarly, for μ_1, we have

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

For Σ, we have

$$\ell(\Sigma) = -\frac{n}{2} \log |\Sigma| - \frac{1}{2} \sum_{i=1}^n \left(x^{(i)} - \mu_{y^{(i)}}\right)^\top \Sigma^{-1} \left(x^{(i)} - \mu_{y^{(i)}}\right) + \text{const}$$

The quadratic term can be rewritten as a trace, so

$$\ell(\Sigma) = -\frac{n}{2} \log |\Sigma| - \frac{1}{2} \sum_{i=1}^n \text{tr}(\Sigma^{-1} \left(x^{(i)} - \mu_{y^{(i)}}\right) \left(x^{(i)} - \mu_{y^{(i)}}\right)^\top)$$

$$\frac{\partial \ell}{\partial \Sigma} = -\frac{n}{2} \Sigma^{-1} + \frac{1}{2} \sum_{i=1}^n \Sigma^{-1} \left(x^{(i)} - \mu_{y^{(i)}}\right) \left(x^{(i)} - \mu_{y^{(i)}}\right)^\top \Sigma^{-1}.$$

Setting this to zero,

$$\boxed{\Sigma = \frac{1}{n} \sum_{i=1}^n \left(x^{(i)} - \mu_{y^{(i)}}\right) \left(x^{(i)} - \mu_{y^{(i)}}\right)^\top}$$

(c) [10 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.

Include two plots of the **validation data** for both datasets 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 figures, plot the decision boundary found by GDA (i.e, line corresponding to $p(y|x) = 0.5$).

Note that your code should be run inside the `src/linearclass/` directory.

Answer: ?? and ??

(d) [2 points] For both datasets, compare the validation set plots obtained in part (c) and PSET 1, Problem 3, part (b) from GDA and logistic regression respectively, and briefly comment on your observation in a couple of lines. On which dataset does GDA seem to perform worse than logistic regression? Why might this be the case?

Answer: Dataset 1, because

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

Answer:

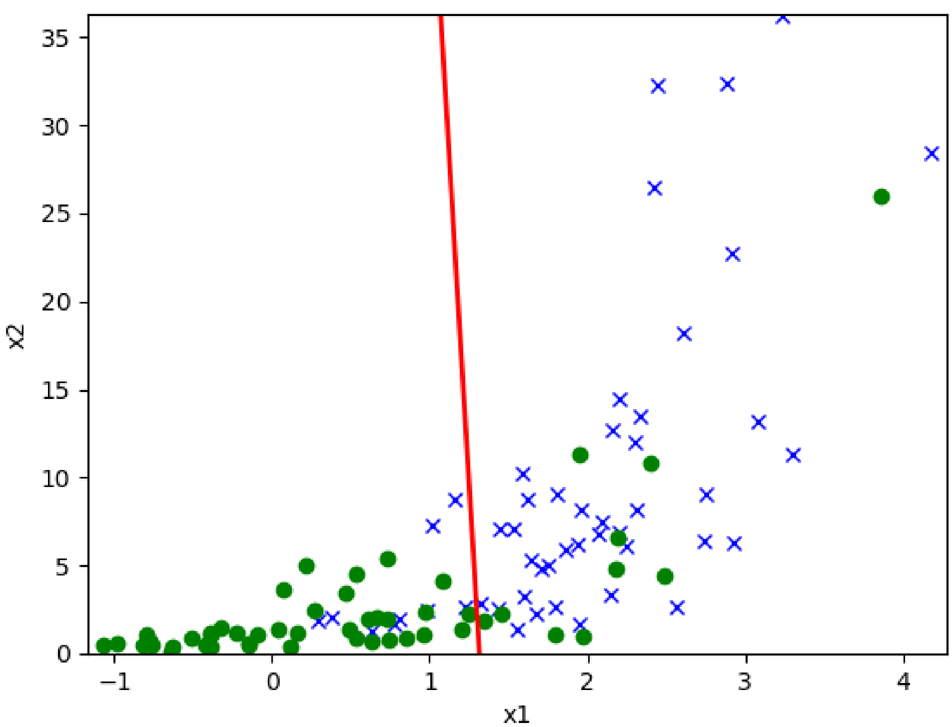

Figure 1: True Counts vs Predicted Counts

3. **[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.

Note that your code should be run inside the `src/spam/` directory.

(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/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 vocabulary size after the pre-processing step. You do not need to include any other output for this subquestion.

Answer:

(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:

(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:

4. **[15 points] Constructing kernels**

In class, we saw that by choosing a kernel $K(x, z) = \phi(x)^T \phi(z)$, we can implicitly map data to a high dimensional space, and have a learning algorithm (e.g., SVM or logistic regression) work in that space. One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding K.

However, in this question, we are interested in direct construction of kernels. I.e., suppose we have a function $K(x, z)$ that we think gives an appropriate similarity measure for our learning problem, and we are considering plugging K into the SVM as the kernel function. However, for $K(x, z)$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. Mercer's theorem tells us that $K(x, z)$ is a (Mercer) kernel if and only if for any finite set $\{x^{(1)}, \ldots, x^{(n)}\}$, the square matrix $K \in \mathbb{R}^{n \times n}$ whose entries are given by $K_{ij} = K(x^{(i)}, x^{(j)})$ is symmetric and positive semidefinite.

(You can find more details about Mercer's theorem in the notes, though the description above is sufficient for this problem.)

In this question we are interested to see which operations preserve the validity of kernels.

Let K_1, K_2 be kernels over $\mathbb{R}^d \times \mathbb{R}^d$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^d \mapsto \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^d \mapsto \mathbb{R}^p$ be a function mapping from $\mathbb{R}^d$ to $\mathbb{R}^p$, and let $p(x)$ a polynomial over x with *positive* coefficients.

For each of the functions K below, state whether it is necessarily a kernel. If you think it is, prove it; if you think it isn't, give a counter-example.

(a) [1 points] $K(x, z) = K_1(x, z) + K_2(x, z)$

(b) [1 points] $K(x, z) = K_1(x, z) - K_2(x, z)$

(c) [1 points] $K(x, z) = aK_1(x, z)$

(d) [1 points] $K(x, z) = -aK_1(x, z)$

(e) [5 points] $K(x, z) = K_1(x, z)K_2(x, z)$

(f) [3 points] $K(x, z) = f(x)f(z)$

(g) [3 points] $K(x, z) = p(K_1(x, z))$

Answer:

5. **[20 points] K-means for compression**

In this problem, we will apply the K-means algorithm to lossy image compression, by reducing the number of colors used in an image.

We will be using the files `src/k_means/peppers-small.tiff` and `src/k_means/peppers-large.tiff`.

The `peppers-large.tiff` file contains a 512×512 image of peppers represented in 24-bit color. This means that, for each of the 262,144 pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The straightforward representation of this image therefore takes about $262144 \times 3 = 786432$ bytes (a byte being 8 bits). To compress the image, we will use K-means to reduce the image to $k = 16$ colors. More specifically, each pixel in the image is considered a point in the three-dimensional (r, g, b)-space. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster centroid.

Follow the instructions below. Be warned that some of these operations can take a while (several minutes even on a fast computer)!

Note that your code should be run inside the `src/k_means/` directory.

(a) [15 points] **[Coding Problem] K-Means Compression Implementation.** First let us *look* at our data. From the `src/k_means/` directory, open an interactive Python prompt, and type

```
from matplotlib.image import imread; import matplotlib.pyplot as plt;
```

and run `A = imread('peppers-large.tiff')`. Now, `A` is a "three dimensional matrix," and `A[:,:,0]`, `A[:,:,1]` and `A[:,:,2]` are 512×512 arrays that respectively contain the red, green, and blue values for each pixel. Enter `plt.imshow(A); plt.show()` to display the image.

Since the large image has 262,144 pixels and would take a while to cluster, we will instead run vector quantization on a smaller image. Repeat (a) with `peppers-small.tiff`.

Next we will implement image compression in the file `src/k_means/k_means.py` which has some starter code. Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, implement K-means with 16 clusters on the pixel data from this smaller image, iterating (preferably) to convergence, but in no case for less than 30 iterations. For initialization, set each cluster centroid to the (r, g, b)-values of a randomly chosen pixel in the image.

Take the image of `peppers-large.tiff`, and replace each pixel's (r, g, b) values with the value of the closest cluster centroid from the set of centroids computed with `peppers-small.tiff`. Visually compare it to the original image to verify that your implementation is reasonable. **Include in your write-up a copy of this compressed image alongside the original image.**

Answer:

(b) [5 points] **Compression Factor.**

If we represent the image with these reduced (16) colors, by (approximately) what factor have we compressed the image? You can ignore the bits required to store the 16 color values themselves.

Answer: