CS 229, Spring 2020
Problem Set #1

Due Wednesday, April 29 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 Piazza forum, at `https://piazza.com/stanford/spring2020/cs229`. (3) This quarter, Spring 2020, students may submit in pairs. If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on the course website before starting work. (4) For the coding problems, you may not use any libraries except those defined in the provided `environment.yml` file. In particular, ML-specific libraries such as scikit-learn are not permitted. (5) To account for late days, the due date is Wednesday, April 29 at 11:59 pm. If you submit after Wednesday, April 29 at 11:59 pm, you will begin consuming your late days. If you wish to submit on time, submit before Wednesday, April 29 at 11:59 pm.

All students must submit an electronic PDF version of the written questions. 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.

1. **[40 points] Linear Classifiers (logistic regression and GDA)**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The log-likelihood of the data is

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

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

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

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

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

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

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

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

(a) First, we can write:

$$\frac{\partial}{\partial \theta_K} h_\theta(x^{(i)}) = \frac{\partial}{\partial \theta_K} g(\theta^T x^{(i)}) = h(x^{(i)})(1-h_\theta(x^{(i)}))x_K. \quad (1)$$

And from (1), we got.

$$\frac{\partial}{\partial \theta_K} \log(h_\theta(x^{(i)})) = \frac{1}{h_\theta(x^{(i)})} \cdot h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x_K^{(i)} = (1-h_\theta(x^{(i)}))x_K^{(i)} \quad (2)$$

$$\frac{\partial}{\partial \theta_K} \log(1-h_\theta(x^{(i)})) = \frac{1}{1-h_\theta(x^{(i)})} \cdot (-h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x_K^{(i)}) = -h_\theta(x^{(i)})x_K^{(i)} \quad (3)$$

Using (1) (2) (3), we calculate the first derivative of $J(\theta)$.

$$\therefore \frac{\partial}{\partial \theta_K} J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}(1-h_\theta(x^{(i)}))x_K^{(i)} - (1-y^{(i)})h_\theta(x^{(i)})x_K^{(i)}\right]$$
$$= -\frac{1}{n}\sum_{i=1}^{n}(y^{(i)}-h_\theta(x^{(i)}))x_K^{(i)}$$

And the second derivative:

$$\therefore \frac{\partial^2}{\partial \theta_K \theta_L} J(\theta) = \frac{\partial}{\partial \theta_L}\left[-\frac{1}{n}\sum_{i=1}^{n}(y^{(i)}-h_\theta(x^{(i)}))x_K^{(i)}\right]$$
$$= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \theta_L} h_\theta(x^{(i)})x_K^{(i)}$$
$$= \frac{1}{n}\sum_{i=1}^{n} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x_L^{(i)}x_K^{(i)}$$

So the (k,l) entry of the Hessian matrix.

$$H_{kl} = \frac{1}{n}\sum_{i=1}^{n} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x_k^{(i)}x_l^{(i)}$$

And the Hessian matrix can be written as.

$$H = \frac{1}{n}\sum_{i=1}^{n} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x^{(i)}x^{(i)T}$$

Prove $H \succeq 0$. Let $z \in \mathbb{R}^d$ be any vector of dimension d.

$$z^T H z = z^T\left(\frac{1}{n}\sum_{i=1}^{n} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x^{(i)}x^{(i)T}\right)z$$

$$= \frac{1}{n}\sum_{i=1}^{n} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))\left[z^T x^{(i)}x^{(i)T}z\right]$$

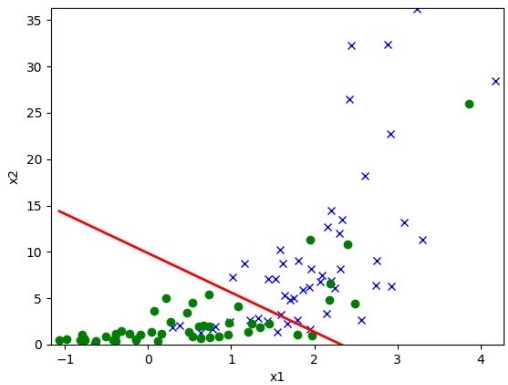
$$= \frac{1}{n}\sum_{i=1}^{n} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))(z^T x^{(i)})^2 \geq 0.$$

Note: ① $h_\theta(x^{(i)})$ is basically a sigmoid function.

$h_\theta(x^{(i)}) \in (0,1)$

② $x^{(i)T}z = (z^T x^{(i)})^T = z^T x^{(i)} \in \mathbb{R}$.

(b)



a). logistic regression on data set 1

(c) $$p(y=1|x;\phi,\Sigma,\mu_0,\mu_1) = \frac{p(x|y=1)\cdot p(y=1)}{p(x)}$$

And $p(x) = p(x|y=1)\cdot p(y=1) + p(x|y=0)\cdot p(y=0)$

$$\begin{cases} p(x|y=1)\cdot p(y=1) = \frac{1}{\sqrt{2\pi}|\Sigma|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right\}\phi \\ p(x|y=0)\cdot p(y=0) = \frac{1}{\sqrt{2\pi}|\Sigma|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right\}(1-\phi). \end{cases}$$

Plug in $p(y=1|x;\phi,\Sigma,\mu_0,\mu_1)$, we get.

$$p(y=1|x;\phi,\Sigma,\mu_0,\mu_1) = \left\{1+\exp\left[\log\frac{1-\phi}{\phi} - \frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) + \frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right]\right\}^{-1}$$

Need to re-write ● as the form of $-(\theta^T x + \theta_0)$

$$\begin{aligned} ● &= \frac{1}{2}\left[x^T\Sigma^{-1}x - 2\mu_1^T\Sigma^{-1}x + \mu_1^T\Sigma^{-1}\mu_1 - (x^T\Sigma^{-1}x - 2\mu_0^T\Sigma^{-1}x + \mu_0^T\Sigma^{-1}\mu_0)\right] + \log\frac{1-\phi}{\phi} \\ &= (\mu_0-\mu_1)^T\Sigma^{-1}x + \frac{1}{2}\left(\mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0\right) + \log\frac{1-\phi}{\phi} \end{aligned}$$

Thus $$\begin{cases} \theta = -\left[(\mu_0-\mu_1)^T\Sigma^{-1}\right]^T = -\Sigma^{-1}(\mu_0-\mu_1) \\ \theta_0 = -\left[\frac{1}{2}\left(\mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0\right) + \log\frac{1-\phi}{\phi}\right] \end{cases}$$

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

$$\begin{aligned} &= \sum_{i=1}^{n}\Big[\log\left(\frac{1}{\sqrt{2\pi}|\Sigma|^{\frac{1}{2}}}\right) + \sum_{i=1}^{n} 1\{y^{(i)}=1\}\left(-\frac{1}{2}(x^{(i)}-\mu_1)^T\Sigma^{-1}(x^{(i)}-\mu_1) + \log\phi\right) \\ &\quad + \sum_{i=1}^{n} 1\{y^{(i)}=0\}\left(-\frac{1}{2}(x^{(i)}-\mu_0)^T\Sigma^{-1}(x^{(i)}-\mu_0) + \log(1-\phi)\right)\Big] \end{aligned}$$

① ϕ:

$$\frac{\partial\ell}{\partial\phi} = \sum_{i=1}^{n} 1\{y^{(i)}=1\}\frac{1}{\phi} + \sum_{i=1}^{n} 1\{y^{(i)}=0\}\left(-\frac{1}{1-\phi}\right)$$

Set $\frac{\partial\ell}{\partial\phi}=0$, Using $n = \sum_{i=1}^{n} 1\{y^{(i)}=1\} + \sum_{i=1}^{n} 1\{y^{(i)}=0\}$

$$\hat{\phi} = \frac{1}{n}1\{y^{(i)}=1\}.$$

② μ_1:

$$\frac{\partial l}{\partial \mu_1} = \sum_{i=1}^{n} \mathbb{1}\{y^{(i)}=1\} \frac{\partial}{\partial \mu_1}\left(x^{(i)T}\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1\right)$$

$$= \sum_{i=1}^{n} \mathbb{1}\{y^{(i)}=1\}\left(\Sigma^{-1}x^{(i)} - \Sigma^{-1}\mu_1\right)$$

Similar to ②, Set $\frac{\partial l}{\partial \mu_1} = 0$. we get

$$\hat{\mu}_1 = \frac{\sum \mathbb{1}\{y^{(i)}=1\} \cdot x^{(i)}}{\sum \mathbb{1}\{y^{(i)}=1\}}.$$

③ μ_0: Similar to μ_1 ②.

④ Σ:

For matrix $A \in \mathbb{R}^{d\times d}$, and vector $z \in \mathbb{R}^d$, we know.

$\frac{\partial |A|}{\partial A} = |A|(A^{-1})^T$ and $\frac{\partial (z^T A z)}{\partial A} = z \cdot z^T$

And imagine if we optimize the log-likelihood for Σ^{-1},

It's the same as Σ. $\frac{\partial l}{\partial \Sigma^{-1}}$ is easier to calculate.

$$\therefore \frac{\partial}{\partial \Sigma^{-1}}\left(-\frac{1}{2}\log|\Sigma|\right) = \frac{\partial}{\partial \Sigma}\left(\frac{1}{2}\log|\Sigma^{-1}|\right) = \frac{1}{2}\Sigma$$

And $\frac{\partial}{\partial \Sigma^{-1}}(x-\mu)^T\Sigma^{-1}(x-\mu) = (x-\mu)^T(x-\mu)$

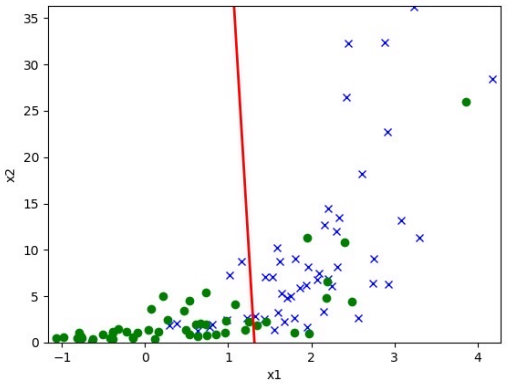
So

$$\frac{\partial l}{\partial \Sigma^{-1}} = \sum_{i=1}^{n}\left(\frac{1}{2}\Sigma - \frac{1}{2}(x^{(i)}-\mu_{y^{(i)}})^T(x^{(i)}-\mu_{y^{(i)}})\right)$$

Set $\frac{\partial l}{\partial \Sigma^{-1}} = 0$.

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

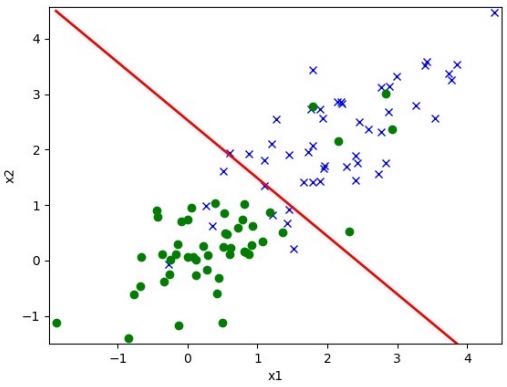
(e)



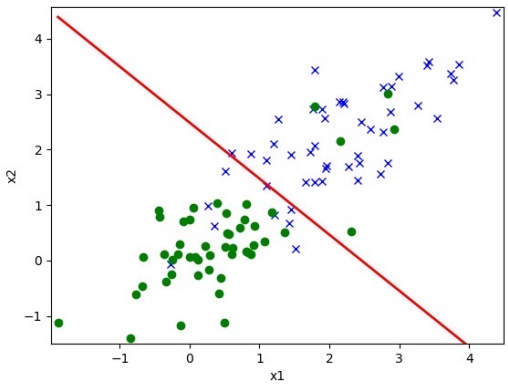
b) gda on data set 1

(f) The decision boundary of logistic regression looks less horizontal than gda's. Logistic regression emphaze on the importance of x_2 more than gda, while, according to gda, change of x_2 doesn't affect the classification much.

(g)



c) logistic regression on data set 2



d) gda on data set 2

(g). On Data Set 1, ADA seems to be worse. The reason might be that $\vec{x}_S$ do not quite follow a multi-variant Gaussian distribution.

(h) x_2 seems to have more value when close to 0, while when x_2 large. Values are sparce. A log transformation would be a reasonable choice for x_2.

2. **[25 points] Poisson Regression**

In this question we will construct another kind of a commonly used GLM, which is called Poisson Regression. In a GLM, the choice of the exponential family distribution is based on the kind of problem at hand. If we are solving a classification problem, then we use an exponential family distribution with support over discrete classes (such as Bernoulli, or Categorical). Simiarly, if the output is real valued, we can use Gaussian or Laplace (both are in the exponential family). Sometimes the desired output is to predict counts, for e.g., predicting the number of emails expected in a day, or the number of customers expected to enter a store in the next hour, etc. based on input features (also called covariates). You may recall that a probability distribution with support over integers (i.e. counts) is the Poisson distribution, and it also happens to be in the exponential family.

In the following sub-problems, we will start by showing that the Poisson distribution is in the exponential family, derive the functional form of the hypothesis, derive the update rules for training models, and finally using the provided dataset train a real model and make predictions on the test set.

(a) [5 points] Consider the Poisson distribution parameterized by λ:

$$p(y; \lambda) = \frac{e^{-\lambda}\lambda^y}{y!}.$$

(Here y has positive integer values and $y!$ is the factorial of y.) Show that the Poisson distribution is in the exponential family, and clearly state the values for $b(y)$, η, $T(y)$, and $a(\eta)$.

(b) [3 points] Consider performing regression using a GLM model with a Poisson response variable. What is the canonical response function for the family? (You may use the fact that a Poisson random variable with parameter λ has mean λ.)

(c) [7 points] For a training set $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, n\}$, let the log-likelihood of an example be $\log p(y^{(i)}|x^{(i)}; \theta)$. By taking the derivative of the log-likelihood with respect to θ_j, derive the stochastic gradient ascent update rule for learning using a GLM model with Poisson responses y and the canonical response function.

(d) [10 points] **Coding problem**

Consider a website that wants to predict its daily traffic. The website owners have collected a dataset of past traffic to their website, along with some features which they think are useful in predicting the number of visitors per day. The dataset is split into train/valid sets and the starter code is provided in the following files:

- `src/poisson/{train,valid}.csv`
- `src/poisson/poisson.py`

We will apply Poisson regression to model the number of visitors per day. Note that applying Poisson regression in particular assumes that the data follows a Poisson distribution whose natural parameter is a linear combination of the input features (*i.e.,* $\eta = \theta^T x$). In `src/poisson/poisson.py`, implement Poisson regression for this dataset and use *full batch gradient ascent* to maximize the log-likelihood of θ. For the stopping criterion, check if the change in parameters has a norm smaller than a small value such as 10^{-5}.

Using the trained model, predict the expected counts for the **validation set**, and create a scatter plot between the true counts vs predicted counts (on the validation set). In the

scatter plot, let x-axis be the true count and y-axis be the corresponding predicted expected count. Note that the true counts are integers while the expected counts are generally real values.

(a) $p(y;\lambda) = \frac{1}{y!}\exp(\ln\lambda \cdot y - \lambda)$

thus $b(y) = \frac{1}{y!}$

$\eta = \ln\lambda$

$T(y) = y$

$a(\eta) = \lambda = e^{\eta}$

(b) Canonical response function is

$g(\eta) = E(T(y);\eta) = E(y;\eta) = \lambda = e^{\eta}$

(c) Note that we assume natural parameter η and $\vec{x}$ are linear related.

$\eta = \theta^T x$. ; $g(\eta) = e^{\eta} = e^{\theta^T x}$.

$$\ell(\theta) = \log p(y^{(i)}|x^{(i)};\theta)$$
$$= \log \frac{e^{-\lambda}\lambda^{y^{(i)}}}{y^{(i)}!}$$
$$= -\ln(y^{(i)}!) - \lambda + y^{(i)}\ln\lambda$$
$$= -\ln(y^{(i)}!) - e^{\theta^T x^{(i)}} + \theta^T x^{(i)} y^{(i)}$$

$$\frac{\partial}{\partial\theta_j}\ell(\theta) = -e^{\theta^T x^{(i)}} \cdot x_j^{(i)} + x_j^{(i)} y^{(i)}$$
$$= (y^{(i)} - e^{\theta^T x^{(i)}}) x_j^{(i)}$$

∴ Update rule

$$\theta_j := \theta_j + \alpha \frac{\partial}{\partial\theta_j}\ell(\theta) = \theta_j + \alpha(y^{(i)} - e^{\theta^T x^{(i)}}) x_j^{(i)}.$$

where α is learning rate

3. **[15 points] Convexity of Generalized Linear Models**

In this question we will explore and show some nice properties of Generalized Linear Models, specifically those related to its use of Exponential Family distributions to model the output.

Most commonly, GLMs are trained by using the negative log-likelihood (NLL) as the loss function. This is mathematically equivalent to Maximum Likelihood Estimation (*i.e.,* maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood). In this problem, our goal is to show that the NLL loss of a GLM is a *convex* function w.r.t the model parameters. As a reminder, this is convenient because a convex function is one for which any local minimum is also a global minimum, and there is extensive research on how to optimize various types of convex functions efficiently with various algorithms such as gradient descent or stochastic gradient descent.

To recap, an exponential family distribution is one whose probability density can be represented

$$p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta)),$$

where η is the *natural parameter* of the distribution. Moreover, in a Generalized Linear Model, η is modeled as $\theta^T x$, where $x \in \mathbb{R}^d$ are the input features of the example, and $\theta \in \mathbb{R}^d$ are learnable parameters. In order to show that the NLL loss is convex for GLMs, we break down the process into sub-parts, and approach them one at a time. Our approach is to show that the second derivative (*i.e.,* Hessian) of the loss w.r.t the model parameters is Positive Semi-Definite (PSD) at all values of the model parameters. We will also show some nice properties of Exponential Family distributions as intermediate steps.

For the sake of convenience we restrict ourselves to the case where η is a scalar. Assume $p(Y|X;\theta) \sim \text{ExponentialFamily}(\eta)$, where $\eta \in \mathbb{R}$ is a scalar, and $T(y) = y$. This makes the exponential family representation take the form

$$p(y;\eta) = b(y)\exp(\eta y - a(\eta)).$$

(a) [5 points] Derive an expression for the mean of the distribution. Show that $\mathbb{E}[Y;\eta] = \frac{\partial}{\partial \eta} a(\eta)$ (note that $\mathbb{E}[Y;\eta] = \mathbb{E}[Y|X;\theta]$ since $\eta = \theta^T x$). In other words, show that the mean of an exponential family distribution is the first derivative of the log-partition function with respect to the natural parameter.

Hint: Start with observing that $\frac{\partial}{\partial \eta}\int p(y;\eta)dy = \int \frac{\partial}{\partial \eta} p(y;\eta)dy$.

(b) [5 points] Next, derive an expression for the variance of the distribution. In particular, show that $\text{Var}(Y;\eta) = \frac{\partial^2}{\partial \eta^2} a(\eta)$ (again, note that $\text{Var}(Y;\eta) = \text{Var}(Y|X;\theta)$). In other words, show that the variance of an exponential family distribution is the second derivative of the log-partition function w.r.t. the natural parameter.

Hint: Building upon the result in the previous sub-problem can simplify the derivation.

(c) [5 points] Finally, write out the loss function $\ell(\theta)$, the NLL of the distribution, as a function of θ. Then, calculate the Hessian of the loss w.r.t θ, and show that it is always PSD. This concludes the proof that NLL loss of GLM is convex.

Hint 1: Use the chain rule of calculus along with the results of the previous parts to simplify your derivations.

Hint 2: Recall that variance of any probability distribution is non-negative.

Remark: The main takeaways from this problem are:

- Any GLM model is convex in its model parameters.

- The exponential family of probability distributions are mathematically nice. Whereas calculating mean and variance of distributions in general involves integrals (hard), surprisingly we can calculate them using derivatives (easy) for exponential family.

(a) We have $\int p(y;\eta)\,dy = 1$ So $\frac{d}{d\eta}\int p(y;\eta)\,dy = \int \frac{d}{d\eta} p(y;\eta)\,dy = 0$

$\therefore \int \frac{d}{d\eta} p(y;\eta)\,dy = \int b(y)\exp(\eta y - a(\eta))(y - \frac{d}{d\eta}a(\eta))\,dy = 0 \quad (p(y;\eta) = f(y;\eta) = \text{pdf of } y)$

$\therefore \underbrace{\int f(y;\eta)\cdot y\cdot dy}_{\text{expectation of } y} = \int f(y;\eta)\underbrace{\frac{d}{d\eta}a(\eta)}_{\text{constant to } dy}\,dy = \frac{d}{d\eta}a(\eta) \Rightarrow E(y;\eta) = \frac{d}{d\eta}a(\eta)$

(b) From (a), we can further derive that $\int \frac{\partial^2}{d\eta^2} p(y;\eta)\,dy = 0$

$\frac{d^2}{d\eta^2}p(y;\eta) = \frac{d}{d\eta}\underline{b(y)(\exp(\eta y - a(\eta))(y - \frac{d}{d\eta}a(\eta))} = y\cdot\frac{d}{d\eta}f(y;\eta) - \frac{d}{d\eta}f(y;\eta)\frac{d}{d\eta}a(\eta) - f(y;\eta)\frac{d^2}{d\eta^2}a(\eta)$

Note that from (a). we have $\frac{d}{d\eta}f(y;\eta) = f(y;\eta)(y - \frac{d}{d\eta}a(\eta))$ and $\frac{d}{d\eta}a(\eta) = E(y)$

$\therefore \int \frac{d^2}{d\eta^2}p(y;\eta)\,dy = \int (y - E(y))(y - E(y))f(y;\eta)\,dy - \frac{d^2}{d\eta^2}a(\eta)\int f(y;\eta)\,dy = 0$

$\Rightarrow \int (y - E(y))^2 f(y)\,dy = \frac{d^2}{d\eta^2}a(\eta) \Rightarrow \frac{d^2}{d\eta^2}a(\eta) = \text{var}(y;\eta)$

(c) Likelihood (for $y = y^{(i)}$).

$L(\theta) = b(y)\exp(\theta^T x\cdot y - a(\theta^T x))$

log-likelihood

$\ell(\theta) = \ln b(y) + \theta^T x\cdot y - a(\theta^T x)$

loss function

$J(\theta) = -\ell(\theta) = a(\theta^T x) - \ln b(y) - \theta^T x\cdot y.$

derivative

$$\frac{\partial}{\partial \theta_j} l(\theta) = \frac{d}{d(\theta^T x)} a(\theta^T x) \cdot \frac{d}{d\theta_j}(\theta^T x) - x_j y$$

$$= E(y) \cdot x_j - x_j y$$

$$= (E(y) - y) x_j$$

Second derivative

$$\frac{\partial^2}{\partial \theta_j \partial \theta_k} l(\theta) = \frac{\partial}{\partial \theta_k} (E(y) - y) x_j$$

$$= \frac{\partial}{\partial \theta_k} \left(\int y \cdot b(y) \cdot \exp(\theta^T x \cdot y - a(\theta^T x)) \, dy \right) x_j - \frac{\partial}{\partial \theta_k} y x_j$$

$$= \int y \cdot b(y) \cdot \exp(\theta^T x \cdot y - a(\theta^T x) \cdot (x_k \cdot y - E(y) \cdot x_k) \, dy \, x_j$$

$$= \int (y - E(y)) \cdot y \cdot f(y) \, dy \, x_j x_k$$

$$= \mathrm{var}(y) \cdot x_k x_j$$

So the Hessian of loss.

$$\nabla^2 l(\theta) = \mathrm{var}(y) \cdot x \cdot x^T$$

Then for any vetor $z \in \mathbb{R}^d$

$$z \cdot \nabla^2 l(\theta) = (z^T x)^2 \cdot \mathrm{var}(y) \geq 0. \quad \Rightarrow \nabla^2 l(\theta) \text{ is PSD}$$

4. **[25 points] Linear regression: linear in what?**

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

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

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

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

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

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

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

In other words, we can use linear regression on the new dataset to find parameters $\theta_0, \ldots, \theta_3$.

Please write down 1) the objective function $J(\theta)$ of the linear regression problem on the new dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$ and 2) the update rule of the batch gradient descent algorithm for linear regression on the dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`src/featuremaps/small.csv`

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

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

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

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

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

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

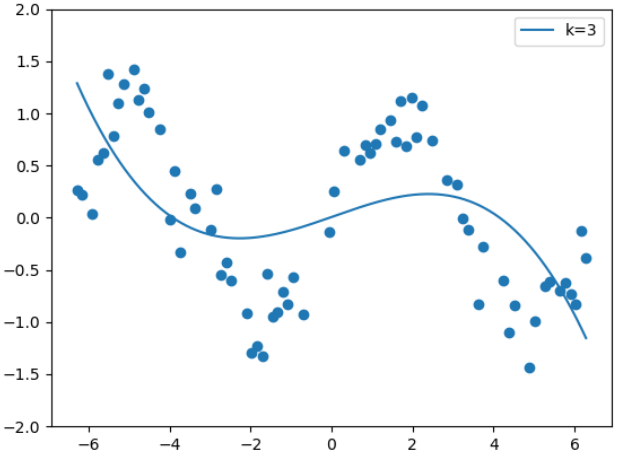
Solution:

(a) 1) $J(\theta) = \frac{1}{2}\sum_{i=1}^{n} (y^{(i)} - \theta^T \hat{x}^{(i)})^2$

2) since $\frac{\partial}{\partial \theta} J(\theta) = \sum_{i=1}^{n} (y^{(i)} - \theta^T \hat{x}^{(i)}) \cdot \hat{x}^{(i)}$

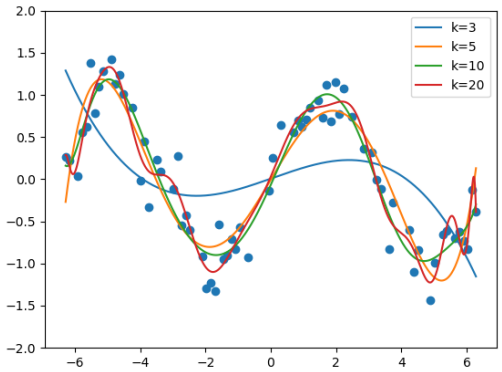
Update rule: $\theta := \theta - \lambda \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})(\hat{x}^{(i)})$

(b)



Learnt hypothesis of degree-3 polynomial

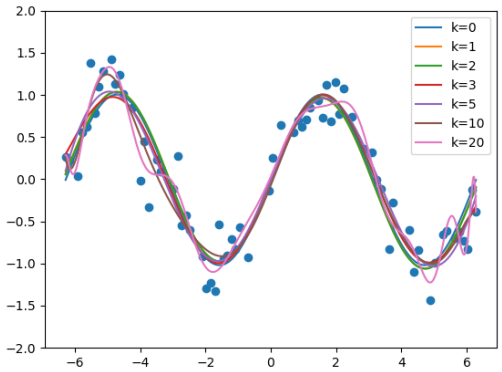
(c)



Learnt curve for $k = [3, 5, 10, 20]$.

Observation: As k becomes large, the curve fits the points better, but also when k = 10 or 20, there are some twists in the curve (variance↑)

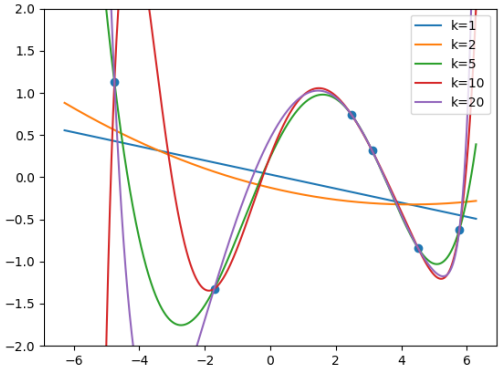
(d)



Learnt curve with $\sin(x)$

Observation: Most curves can better capture the periodical shape while as k becomes large. The curve tries to fit the noise.

(d)



Learnt curve on small data set

Observations: k large, the curves tend to predict extreme values on two sides of the data set, though they have perfect fit for the data points.

5. **[10 points] Linear invariance of optimization algorithms**

Consider using an iterative optimization algorithm (such as Newton's method, or gradient descent) to minimize some continuously differentiable function $f(x)$. Suppose we initialize the algorithm at $x^{(0)} = \vec{0}$. When the algorithm runs, it will produce a value of $x \in \mathbb{R}^d$ for each iteration: $x^{(1)}, x^{(2)}, \ldots$.

Now, let some non-singular square matrix $A \in \mathbb{R}^{d \times d}$ be given, and define a new function $g(z) = f(Az)$. Consider using the same iterative optimization algorithm to optimize g (with initialization $z^{(0)} = \vec{0}$). If the values $z^{(1)}, z^{(2)}, \ldots$ produced by this method necessarily satisfy $z^{(i)} = A^{-1}x^{(i)}$ for all i, we say this optimization algorithm is **invariant to linear reparameterizations**.

(a) [7 points] Show that Newton's method (applied to find the minimum of a function) is invariant to linear reparameterizations. Note that since $z^{(0)} = \vec{0} = A^{-1}x^{(0)}$, it is sufficient to show that if Newton's method applied to $f(x)$ updates $x^{(i)}$ to $x^{(i+1)}$, then Newton's method applied to $g(z)$ will update $z^{(i)} = A^{-1}x^{(i)}$ to $z^{(i+1)} = A^{-1}x^{(i+1)}$.[1]

(b) [3 points] Is gradient descent invariant to linear reparameterizations? Justify your answer.

(a) Recall Newton's method to minimize $f(x)$, the update rule is

$$x^{(i+1)} = x^{(i)} - \frac{\nabla_x f(x^{(i)})}{Hf(x^{(i)})} \quad (1)$$

Need to prove when $z^{(i)} = A^{-1}x^{(i)}$, then $z^{(i+1)}$ obtained by

$$z^{(i+1)} = z^{(i)} - \frac{\nabla_z g(z^{(i)})}{Hg(z^{(i)})} \quad (2)$$

statisfies $z^{(i+1)} = A^{-1}x^{(i+1)}$

$$\nabla_z g(z^{(i)}) = \frac{\partial}{\partial z} f(Az^{(i)}) = A^T \nabla_x f(x^{(i)})$$

$$Hg(z^{(i)}) = \nabla_z^2 g(z^{(i)}) \quad (3)$$

$$H_{ij} = \frac{\partial^2}{\partial z_i \partial z_j} g(z^{(i)}) = \frac{\partial^2}{\partial z_i z_j} f(Az^{(i)}) = \sum_k \sum_l \frac{\partial^2}{\partial x_k \partial x_l} f(x^{(i)}) \cdot A_{ki} A_{lj}.$$

[1] Note that for this problem, you must explicitly prove any matrix calculus identities that you wish to use that are not given in the lecture notes.

$$H_g(z^{(i)}) = A^T H_f(x^{(i)}) A. \quad (4)$$

From (3) and (4), re-write (3) as:

$$z^{(i+1)} = A^{-1}x^{(i)} - \frac{A^T \nabla_x f(x^{(i)})}{A^T H_f(x^{(i)}) A}$$

Recall from (1): $-\frac{\nabla_x f(x^{(i)})}{H_f(x^{(i)})} = x^{(i+1)} - x^{(i)}$

$$\therefore z^{(i+1)} = A^{-1}x^{(i)} - A^{-1}(x^{(i+1)} - x^{(i)})$$

$$= A^{-1}x^{(i+1)}$$

(b) No, Gradient descent is not invariant

Recall update rule for gradient descent.

$$x^{(i+1)} = x^{(i)} - \alpha f'(x^{(i)})$$ → loss function.

Assume that after i iterations, we still have

$$z^{(i)} = A^{-1}x^{(i)},$$

if invariant, we have

$$z^{(i+1)} = A^{-1}x^{(i+1)}$$

$$z^{(i+1)} = z^{(i)} - \alpha g'(z^{(i)})$$

$$= z^{(i)} - \alpha \frac{d}{dz} f(Az^{(i)})$$

$$= z^{(i)} - \alpha \frac{d}{dAz^{(i)}} f(Az^{(i)}) \cdot \frac{d}{dz} Az^{(i)}$$

$$= z^{(i)} - \alpha A f'(x^{(i)})$$

Similarly, plug in $f'(x^{(i)}) = \frac{1}{\alpha}(x^{(i)} - x^{(i+1)})$

then $z^{(i+1)} = A^{-1}x^{(i)} - Ax^{(i)} + Ax^{(i+1)}$

If $A^{-1}x^{(i)} - Ax^{(i)} + Ax^{(i+1)} = A^{-1}x^{(i+1)}$

then $(A^{-1} - A)(x^{(i)} - x^{(i+1)}) = 0$

$A \neq I \Rightarrow A^{-1} \neq A \Rightarrow A^{-1} - A \neq 0$

$\Rightarrow x^{(i)} - x^{(i+1)} = -\alpha f'(x^{(i)}) = 0$

$\Rightarrow f'(x^{(i)}) = 0$

that only happens when $f(x^{(i)}) = \min f(x)$

Contradiction

6. **[25 points] 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.

(a) [10 points] **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) (3 point) 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

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

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

$$\overline{A} \triangleq \frac{1}{2}(A_0 + A_1) \tag{6}$$

With these notations, we can verify that the accuracy is equal to $A = \frac{TP+TN}{TP+TN+FP+FN}$.
(ii) (4 point) Show that

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

and

$$A = \rho \cdot A_1 + (1 - \rho)A_0 \tag{7}$$

Comparing equation (6) and (7), we can see that the accuracy and balanced accuracy are both linear combination of A_0 and A_1 but with different weighting.
(iii) (3 point) 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) [5 points] **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/imbalanced.py`

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 logistical loss, where $h_\theta(x) = g(\theta^T x), g(z) = 1/(1 + e^{-z})$, similar to Problem 1 Part a:

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

You are encouraged to use your code for problem 1, but you are also allowed to use any standard logistic regression library or package to optimize the objective above. 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.
Create a plot to visualize the validation set with x_1 on the horizontal axis and x_2 on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $y^{(i)} = 1$ than those with $y^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5) in red color. Include this plot in your writeup.

(c) [5 points] **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 logistical regression objective for 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) [5 points] **Coding problem: re-weighting minority class**

In `src/imbalanced/imbalanced.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.

Create a plot to visualize the validation set with x_1 on the horizontal axis and x_2 on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $y^{(i)} = 1$ than those with $y^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5) in red color. Include this plot in your writeup.

Solution

(a) (i) Imagine classifier f, $f(x) = 0$ → predict every input as negative.

$$A = \frac{\#\text{ negative example}}{\#\text{ of total example}} = 1 - \rho$$

(ii) # of positive cases = TP + FN.

# of total cases = TP + TN + FP + FN.

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

So $\rho = \frac{\#\text{ of positive cases}}{\#\text{ of total case}} = \frac{TP+FN}{TP+TN+FP+FN}$

$$\rho \cdot A_1 + (1-\rho) A_0 = \frac{TP+FN}{TP+TN+FP+FN} \cdot \frac{TP}{TP+FN} + \left(1 - \frac{TP+FN}{TP+TN+FP+FN}\right) \cdot \frac{TN}{TN+FP}$$

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

A has the weight ρ and $(1-\rho)$ for positive and negative cases while $\bar{A}$ has 0.5 for pos/neg cases.

(c) For any classifier. in D $A_1 = \frac{TP}{TP+FN}$. in D' $A_1' = \frac{TP \times \frac{1}{k}}{(TP+FN)\times\frac{1}{k}} = \frac{TP}{TP+FN}$.

$\therefore \bar{A}_D = \frac{1}{2}(A_0 + A_1) = \frac{1}{2}(A_0' + A_1') = \bar{A}_{D'}$ ✓

In D.

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

In D'. # of pos $= k \times$ # of neg.

$$(\#\text{ of pos})' = \left(\frac{1}{k}\right)(\#\text{ of pos})$$

$$= \#\text{ of neg}$$

$$\therefore (\#\text{ of total case})' = \frac{2}{1+k} n$$

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

in all n' terms. we can see all the terms for negative case remained the same.

But for positive case. e.g. j. there are $\frac{1}{k}$ terms for each one.

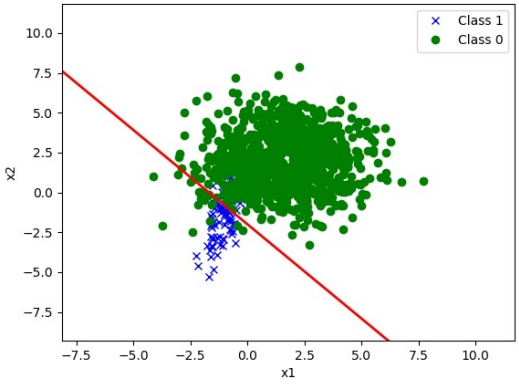
$$\frac{1}{K}\left(y^{(j)}\log(h(x^{(i)})) + (1-y^{(j)})\log(1-h(x^{(i)}))\right)$$

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

$$\begin{cases} w^{(i)} = \frac{1}{K}, & \text{if } y^{(i)} = 1 \\ w^{(i)} = 1, & \text{if } y^{(i)} = 0 \end{cases}$$

Coding questions:

(b)



Naive logistic regression decision boundary.

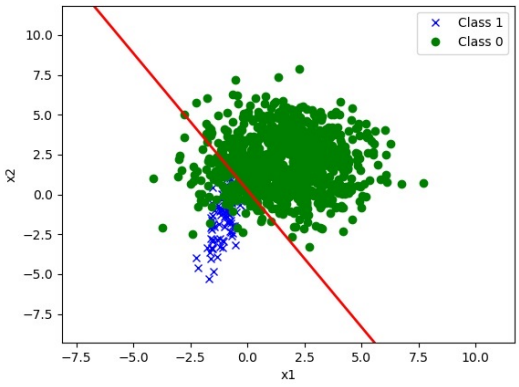
Accuracy A: 94.73%

Balanced Accuracy $\bar{A}$: 77.75%

Positive Accuracy A_1: 57.00%

Negative Accuracy A_0: 98.50%

(d)



Upsampling Decision Boundary

Accuracy A: 89.91%

Balanced Accuracy $\bar{A}$: 90.40%

Positive Accuracy A_1: 91.00%

Negative Accuracy A_0: 89.80%