CP8318/CPS803: Machine Learning (Fall 2021)
Assignment 2
Due: Monday November 8, 2021 (midnight 11:59PM or 23:59)
Instructor: Nariman Farsad

Please Read Carefully:

- Submit your assignments on D2L before the deadline. The submission will be closed after the deadline.
- Show your work and write your solutions legibly. You can use applications such as Word or Latex to type up your solutions for the written portion of the assignment or use applications such as CamScanner to create a PDF file from your handwritten solutions. For coding problems you need to submit your **python source files in the format given**, and have the main plots and discussion in the PDF written portion of the solution.
- Do the assignment on your your own, i.e., do not copy. If two assignments are found to be copies of each other, they BOTH receive 0.
- These questions require thought, but do not require long answers. Please be as concise as possible.
- If you have a question about this homework, we encourage you to post your question on our D2L discussion board.
- Students in CP8318 are required to answer all questions in the assignment. CPS803 students are not required to answer the questions marked for CP8318 students. These questions will not be graded for CPS803 students and they can just attempt them for practice.
- For instructions on setting up the python environment for the assignments read the "README.md" file.
- 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.

1. (30 points) **Logistic regression and GDA**

   We compare and contrast two linear classifiers in this problem: logistic regression, and 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.

   1. [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 $p(y|x) = 0.5$). The function `plot` in `src/linearclass/utils.py` already does this for you. You just need to call it with the right inputs.

   2. [10 points] **Coding problem.** Recall that in GDA we model the joint distribution of (x, y) by the following equations:

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

      where ϕ, μ_0, μ_1, and Σ are the parameters of our model, and $p(x, y) = p(x|y)p(y)$.

      Recall that given the dataset, the maximum likelihood estimates of the parameters of GDA are

given by

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

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

$$p(y = 1 \mid x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-(\theta^T x + \theta_0))}, \tag{0.1}$$

where $\theta \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$ are appropriate functions of ϕ, Σ, μ_0, and μ_1. The θ_0 and θ are given by

$$\theta_0 = \frac{1}{2}(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1) - \log\frac{1-\phi}{\phi}, \tag{0.2}$$
$$\theta = -\Sigma^{-1}(\mu_0 - \mu_1). \tag{0.3}$$

For students looking for extra challenge, try and prove equations (0.1)–(0.3). No extra credit will be given if you do and it will not be graded.

In `src/linearclass/gda.py`, fill in the code for the `fit` function to calculate ϕ, μ_0, μ_1, and Σ, and use these parameters to evaluate θ_0 and θ using Equations (0.2)-(0.3). Note that you can combine θ_0 and θ into $\Theta = [\theta_0, \theta]$ by concatenating θ_0 with θ. Now defining $x_0 = 1$, you can use the resulting GDA model parameterized with Θ to make predictions on the validation set using Equation (0.1) above, which is the same as the perdition in logistic regression. 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 $p(y|x) = 0.5$). The function `plot` in `src/linearclass/utils.py` already does this for you. You just need to call it with the correct inputs.

3. [5 points] For Dataset 1, compare the validation set plots obtained in part (1.1) and part (1.2) from logistic regression and GDA respectively, and briefly comment on your observation in a couple of lines.

4. [5 points] Repeat the process 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?

5. [5 points] **CP8318 Only Question:** For the dataset where GDA performed worse in, can you find a transformation of the $x^{(i)}$'s such that GDA performs significantly better? What might this transformation be?

2. (20 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, and finally using the provided dataset train a real model and make predictions on the test set.

1. [5 points] **CP8318 Only Question:** 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 $a(\eta)$. [**Hint**: Note that you can use the fact that $a = \exp(\log a)$]

2. [5 points] **CP8318 Only Question:** Consider performing regression using a GLM model with a Poisson response variable. Show that the canonical response function is $g(\eta) = \mathbb{E}[y; \eta] = e^{\eta} = e^{\theta^T x}$. (You may use the fact that a Poisson random variable with parameter λ has mean λ, and that $g(\eta) = \mathbb{E}[y; \eta]$ by definition.)

3. [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}. Note that using the canonical response function we have that for Poisson regression $h_\theta(x^{(i)}) = e^{\theta^T x^{(i)}}$, and the the update rule for gradient descent over the cost function given my the negative of the log-likelihood is given by:

$$\theta_j = \theta_j - \alpha(e^{\theta^T x^{(i)}} - y^{(i)})x_j^{(i)}.$$

You can use the same technique as in assignment 1 to implement gradient descent. For this question you don't need to normalize the features, and use $\alpha = 10^{-5}$ (in the code α is called the `step_size` or `lr`).

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. Briefly explain what you observe in the plot.

3. (40 points) **Spam classification**

   In this problem, we will use the naive Bayes algorithm and an SVM 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` [1]

   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 between the spam and non-spam messages using the text of the SMS message.

   1. [10 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 `src/spam/spam_dictionary` and a sample of the resulting training matrix into `src/spam/spam_sample_train_matrix`.

   2. [10 points] In this question you are going to implement a naive Bayes classifier for spam classification with multinomial event model and Laplace smoothing (refer to class notes on Naive Bayes for details on Laplace smoothing).

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

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

[1] 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.

Remark. If you implement naive Bayes the straightforward way, you'll 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.]

3. [10 points] Intuitively, some tokens (i.e., words in our dictionary) 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:

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

   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.

   The provided code will print out the resulting indicative tokens and then save them to `src/spam/top_indicative_words`.

4. [10 points] Support vector machines (SVMs) are an alternative machine learning model that we discussed in class. We have provided you an SVM implementation (using a radial basis function (RBF) kernel) within `src/spam/svm.py` (You should not need to modify that code).

   One important part of training an SVM parameterized by an RBF kernel is choosing an appropriate kernel radius.

   Complete the `compute_best_svm_radius` by writing code to compute the best SVM radius which maximizes accuracy on the validation dataset.

   The provided code will use your `compute_best_svm_radius` to compute and then write the best radius into `src/spam/optimal_radius`. Compare the performance of the SVM with Naive Bayes. [**Hint:** The method `train_and_predict_svm` in `src/spam/svm.py` will be useful.]