Georgia Institute of Technology

ECE 6254 - Assignment 6

Fall 2020 - v1.2

- There are 2 problems over 4 pages (including the cover page)
- The problems are not necessarily in order of difficulty.
- All problems are assigned the same weight in the overall grade.
- Each question is graded out of two points (0 for no meaningful work, 1 for partial work, 2 if correct)
- Unless otherwise specified, you should concisely indicate your reasoning and show all relevant work.
- The grade on each problem is based on our judgment of your level of understanding as reflected by what you have written. If we can't read it, we can't grade it.
- Please use a pen and not a pencil if you handwrite your solution.
- **You must submit your exam on Gradescope. Make sure you allocate time for the submission**.

Problem 1: Gaussian and Uniform Naive Bayes

Consider Gaussian Naive Bayes classifiers for a dataset with a single feature x and two classes 0 and 1. We will classify a point x as class 1 if

$$P(y = 1|x) \geqslant P(y = 0|x).$$

In class we have seen that for certain models of the generative distributions $P(x|y)$, the above decision rule gives rise to a *linear* decision boundary. In the following, you will be given different models, and you will write down the expression for the corresponding decision boundaries. We use $\mathcal{N}(\mu, \sigma^2)$ to denote a normal distribution with mean μ and variance σ^2. (*Hint:* You should be able to figure out the answer by sketching the distributions).

[Q1] Our first model is $x|y = 0 \sim \mathcal{N}(0, 1)$, $x|y = 1 \sim \mathcal{N}(2, 1)$ and $P(y = 1) = 0.5$. Is the decision boundary of this classifier linear? In other words, can you write a linear expression of the form $w_0 + w_1 x \geqslant 0$ to represent the decision boundary of this model? If yes, please give the values of w_0 and w_1. If no, please explain why not.

[Q2] Our second model is $x|y = 0 \sim \mathcal{N}(0, \frac{1}{4})$, $x|y = 1 \sim \mathcal{N}(0, 1)$ and $P(y = 1) = 0.5$. Is the decision boundary of this second model linear? If yes, please check the right decision boundary from the following options and give a brief explanation.

- Classify as class 1 if $x \geqslant 1/2$.
- Classify as class 1 if $x \leqslant -1/2$.
- Classify as class 1 if $x \leqslant 1$.
- Classify as class 1 if $x \geqslant -1$.
- Decision boundary is not linear.

[Q3] Now consider a quadratic decision boundary. In a quadratic decision boundary we add a new feature x^2 for a data-point (x, y). Thus we convert every data-point (x, y) (x^2, x, y). A linear decision boundary for this modified dataset yields a quadratic boundary for the original dataset. Is it possible to find a quadratic decision boundary which matches the decision boundary of the second model exactly? Please check the right decision boundary from the following options and give a brief explanation.

- Classify as class 1 if $x \geqslant \sqrt{\frac{2}{3}\log 2}$ or $x \leqslant -\sqrt{\frac{2}{3}\log 2}$
- Classify as class 1 if $x \leqslant \sqrt{\frac{2}{3}\log 2}$ and $x \geqslant -\sqrt{\frac{2}{3}\log 2}$
- Decision boundary is not quadratic.

Suppose now that you are building a Naive Bayes classifier for data containing two continuous attributes, x_1 and x_2, in addition to the label $y \in \{y_1, \cdots, y_K\}$. You decide to represent the conditional distributions $p(x_i|y)$ as uniform distributions; that is, to build a Uniform Naive Bayes Classifier. Your conditional distributions would be represented using the parameters $a_{i,k} < b_{i,k}$:

$$p(x_i|y = y_k) = \begin{cases} \frac{1}{b_{i,k} - a_{i,k}} & \text{if } a_{i,k} \leqslant x_i \leqslant b_{i,k} \\ 0 & \text{else.} \end{cases}$$

Note: the index i refers here to the component (1 or 2) of the vector, not to the index of a point in the dataset.

[Q4] Given a set of data, how would we choose MLEs for the parameters $a_{i,k}$ and $b_{i,k}$?

[Q5] Assuming that you have formed estimates of $a_{i,k}$, $b_{i,k}$, and the priors, what will the decision surface look like? Provide a sketch to illustrate your answer.

Problem 2: Solvers for logistic regression

In this problem, you will explore implementations of three different solvers for logistic regression. To get started, download the file `Gradient descent.ipynb` from Canvas.

[Q1] We will begin by implementing logistic regression using standard gradient descent for performing the maximum likelihood estimation step. Review the slides and lecture notes and make sure you understand what `Gradient descent.ipynb` is doing. Test out the file by experimenting a bit with the various parameters – especially the "step size" (or learning rate) α – to obtain a good convergence rate. I would recommend trying a wide variety starting with $\alpha \approx 1 \cdot 10^{-3}$ and ranging to $\alpha \approx 1 \cdot 10^{-7}$. You should also feel free to play around with other stopping criteria than those provided. For this problem, all you need to do is to report the value of α you used, the number of iterations required for convergence for these parameters, and a plot showing the "cost" decreasing with iterations.

[Q2] We now want to implement Newton's method. This algorithm requires computing both the gradient and the Hessian at each iteration. Show that the Hessian in this case is given by

$$\nabla^2_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) = -\sum_i \mathbf{x}_i \mathbf{x}_i^\mathsf{T} \frac{1}{1+e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}_i}} \left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}_i}}\right). \tag{1}$$

Be careful: Note that we are using the convention that the first component of $\mathbf{x}_i$ is equal to one, to lump $\mathbf{w}$ and b into a single vector $\boldsymbol{\theta}$.

[Q3] Adapt `Gradient descent.ipynb` to implement Newton's method for solving this problem. You should complete this assignment by writing a new function to replace `grad_desc`. Submit the code you produce for this function (do not include the plotting commands or any functions provided in `Gradient descent.ipynb`), and report the number of iterations required for convergence. Compare this to the results from part **(a)**. Pay attention to the following:

- Take extra care in your implementation in making sure you have the correct signs in your algorithm – remember we are trying to minimize the negative log-likelihood. If your algorithm is not converging, this is a likely suspect.
- In the notes I assume that the $\mathbf{x}_i$ are column vectors, but in the Python code they are treated as row vectors.
- Make sure you use the proper operations in python. In particular `A*B` is not a matrix multiplication.

[Q4] We will now implement yet another variant of gradient descent called *stochastic gradient descent* to perform the optimization step in logistic regression. In standard gradient descent, in order to compute the gradient we must compute the sum

$$\sum_{i=1}^{N} \mathbf{x}_i \left(y_i - \frac{1}{1+e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}_i}}\right) \tag{2}$$

In the data set we're using here, this is not much of a challenge, but if our data set is extremely massive (i.e., if N is extraordinarily large) then even simply computing the gradient can be computationally intractable. One possibility in this case is known as stochastic gradient descent, and consists of simply selecting one $\mathbf{x}_i$ at random and treating $\mathbf{x}_i \left(y_i - \frac{1}{1+e^{-\theta^T x_i}} \right)$ as a rough approximation to (2) and proceeding with the standard gradient descent approach as before. This will (in general) require more iterations, but each iteration can be much cheaper when n is large, so can be very effective when dealing with extremely large data sets. Implement a version of stochastic gradient descent by writing a new function to replace `grad_desc` in `Gradient descent.ipynb`. Submit your code with your assignment, and report the number of iterations required for convergence. Compare this to the results from part **(a)**. You may find the command `np.random.permutation(N)` to be of use to go through the points of the dataset in a random order.

[Q5] Finally, try comparing the results from parts **(a)**, **(b)**, and **(c)** when applied to a much larger dataset. Try changing `n_samples` to $100,000$. You will want to comment out the plotting portion of the code for this part (or at least the part that plots the training data). Record both the running time and the number of iterations for all three algorithms when applied to this dataset. Note that in order to run traditional gradient descent on this large of a dataset, you will need to make α much smaller (think about why this would be the case). If you see warnings regarding a divide by zero, you have set α to be too large. Also, note that you can get the system time in Python by adding `import time` to your file and then using the `time.time()` command.