Homework 3

Deadline: Wednesday, Nov. 4, at 11:59pm.

Submission: You need to submit the following files through MarkUs[1]:

- Your answers to Questions 1, 2, and 3, and outputs requested for Question 2 and 3, as a PDF file titled `hw3_writeup.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.
- Python file `naive_bayes.py` completed for Questions 2 and 3.

Neatness Point: One point will be given for neatness. You will receive this point as long as we don't have a hard time reading your solutions or understanding the structure of your code.

Late Submission: 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.

Computing: To install Python and required libraries, see the instructions on the course web page.

Homeworks are individual work. See the Course Information handout[2] for detailed policies.

1. **[7pts] AdaBoost.**

   (a) **[5pts]** The goal of this question is to show that the AdaBoost algorithm changes the weights in order to force the weak learner to focus on difficult data points. Here we consider the case that the target labels are from the set $\{-1, +1\}$ and the weak learner also returns a classifier whose outputs belongs to $\{-1, +1\}$ (instead of $\{0, 1\}$). Consider the t-th iteration of AdaBoost, where the weak learner is

$$h_t \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\},$$

   the w-weighted classification error is

$$\text{err}_t = \frac{\sum_{i=1}^N w_i \mathbb{I}\{h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^N w_i},$$

   and the classifier coefficient is $\alpha_t = \frac{1}{2} \log \frac{1-\text{err}_t}{\text{err}_t}$. (Here, log denotes the natural logarithm.) AdaBoost changes the weights of each sample depending on whether the weak learner h_t classifies it correctly or incorrectly. The updated weights for sample i is denoted by w_i' and is

$$w_i' \leftarrow w_i \exp\Big(-\alpha_t t^{(i)} h_t(\mathbf{x}^{(i)})\Big).$$

   Show that the error w.r.t. $(w_1', \ldots, w_N')$ is exactly $\frac{1}{2}$. That is, show that

$$\text{err}_t' = \frac{\sum_{i=1}^N w_i' \mathbb{I}\{h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^N w_i'} = \frac{1}{2}.$$

[1] https://markus.teach.cs.toronto.edu/csc311-2020-09

[2] http://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/syllabus.pdf

Note that here we use the weak learner of iteration t and evaluate it according to the new weights, which will be used to learn the $t+1$-st weak learner. What is the interpretation of this result?

Hints:

i. Start from err'_t and divide the summation to two sets of $E = \{i : h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}$ and its complement $E^c = \{i : h_t(\mathbf{x}^{(i)}) = t^{(i)}\}$.

ii. Note that
$$\frac{\sum_{i \in E} w_i}{\sum_{i=1}^{N} w_i} = \text{err}_t.$$

(b) **[2pts]** Recall from lecture that we can rewrite the 0-1 loss as: $\mathbb{I}[h(x^{(n)}) \neq t^{(n)}] = \frac{1}{2}(1 - h(x^{(n)}) \cdot t^{(n)})$. Use this identity to prove that the weight update from part (a):

$$w'_i \leftarrow w_i \exp\Big(-\alpha_t t^{(i)} h_t(\mathbf{x}^{(i)})\Big).$$

is proportional, up to a constant factor, to weight update:

$$w'_i \leftarrow w_i \exp\Big(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}\Big).$$

What is the constant factor? Since we normalize the weights when computing err_t, these two updates are equivalent.

2. **[13pts] Fitting a Naïve Bayes Model**

In this question, we'll fit a Naïve Bayes model to the MNIST digits using maximum likelihood. The starter code will download the dataset and parse it for you: Each training sample $(\mathbf{t}^{(i)}, \mathbf{x}^{(i)})$ is composed of a vectorized binary image $\mathbf{x}^{(i)} \in \{0,1\}^{784}$, and 1-of-10 encoded class label $\mathbf{t}^{(i)}$, i.e., $t_c^{(i)} = 1$ means image i belongs to class c.

Given parameters $\boldsymbol{\pi}$ and $\boldsymbol{\theta}$, Naïve Bayes defines the joint probability of the each data point $\mathbf{x}$ and its class label c as follows:

$$p(\mathbf{x}, c \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = p(c \mid \boldsymbol{\theta}, \boldsymbol{\pi}) p(\mathbf{x} \mid c, \boldsymbol{\theta}, \boldsymbol{\pi}) = p(c \mid \boldsymbol{\pi}) \prod_{j=1}^{784} p(x_j \mid c, \theta_{jc}).$$

where $p(c \mid \boldsymbol{\pi}) = \pi_c$ and $p(x_j = 1 \mid c, \boldsymbol{\theta}, \boldsymbol{\pi}) = \theta_{jc}$. Here, $\boldsymbol{\theta}$ is a matrix of probabilities for each pixel and each class, so its dimensions are 784×10, and $\boldsymbol{\pi}$ is a vector with one entry for each class. (Note that in the lecture, we simplified notation and didn't write the probabilities conditioned on the parameters, i.e. $p(c|\boldsymbol{\pi})$ is written as $p(c)$ in lecture slides).

For binary data ($x_j \in \{0,1\}$), we can write the Bernoulli likelihood as

$$p(x_j \mid c, \theta_{jc}) = \theta_{jc}^{x_j} (1 - \theta_{jc})^{(1-x_j)}, \tag{0.1}$$

which is just a way of expressing $p(x_j = 1 | c, \theta_{jc}) = \theta_{jc}$ and $p(x_j = 0 | c, \theta_{jc}) = 1 - \theta_{jc}$ in a compact form. For the prior $p(\mathbf{t} \mid \boldsymbol{\pi})$, we use a categorical distribution (generalization of Bernoulli distribution to multi-class case),

$$p(t_c = 1 \mid \boldsymbol{\pi}) = p(c \mid \boldsymbol{\pi}) = \pi_c \quad \text{or equivalently} \quad p(\mathbf{t} \mid \boldsymbol{\pi}) = \Pi_{j=0}^{9} \pi_j^{t_j} \quad \text{where} \quad \sum_{i=0}^{9} \pi_i = 1,$$

where $p(c \mid \boldsymbol{\pi})$ and $p(\mathbf{t} \mid \boldsymbol{\pi})$ can be used interchangeably. You will fit the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ using MLE and MAP techniques. In both cases, your fitting procedure can be written as a few simple matrix multiplication operations.

(a) **[3pts]** First, derive the *maximum likelihood estimator* (MLE) for the class-conditional pixel probabilities $\boldsymbol{\theta}$ and the prior $\boldsymbol{\pi}$. Derivations should be rigorous.

Hint 1: We saw in lecture that MLE can be thought of as 'ratio of counts' for the data, so what should $\hat{\theta}_{jc}$ be counting?

Hint 2: Similar to the binary case, when calculating the MLE for π_j for $j = 0, 1, ..., 8$, write $p(\boldsymbol{t}^{(i)} \mid \boldsymbol{\pi}) = \Pi_{j=0}^{9} \pi_j^{t_j^{(i)}}$ and in the log-likelihood replace $\pi_9 = 1 - \Sigma_{j=0}^{8} \pi_j$, and then take derivatives w.r.t. π_j. This will give you the ratio $\hat{\pi}_j / \hat{\pi}_9$ for $j = 0, 1, .., 8$. You know that $\hat{\pi}_j$'s sum up to 1.

(b) **[1pt]** Derive the log-likelihood $\log p(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\pi})$ for a single training image.

(c) **[3pt]** Fit the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ using the training set with MLE, and try to report the average log-likelihood per data point $\frac{1}{N} \Sigma_{i=1}^{N} \log p(\mathbf{t}^{(i)}|\mathbf{x}^{(i)}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\pi}})$, using Equation (0.1). What goes wrong? (it's okay if you can't compute the average log-likelihood here).

(d) **[1pt]** Plot the MLE estimator $\hat{\boldsymbol{\theta}}$ as 10 separate greyscale images, one for each class.

(e) **[2pt]** Derive the *Maximum A posteriori Probability* (MAP) estimator for the class-conditional pixel probabilities $\boldsymbol{\theta}$, using a Beta(3, 3) prior on each θ_{jc}. Hint: it has a simple final form, and you can ignore the Beta normalizing constant.

(f) **[2pt]** Fit the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ using the training set with MAP estimators from previous part, and report both the average log-likelihood per data point, $\frac{1}{N} \Sigma_{i=1}^{N} \log p(\mathbf{t}^{(i)}|\mathbf{x}^{(i)}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\pi}})$, and the accuracy on both the training and test set. The accuracy is defined as the fraction of examples where the true class is correctly predicted using $\hat{c} = \text{argmax}_c \log p(t_c = 1|\mathbf{x}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\pi}})$.

(g) **[1pt]** Plot the MAP estimator $\hat{\boldsymbol{\theta}}$ as 10 separate greyscale images, one for each class.

3. **[4pts] Generating from a Naïve Bayes Model**

Defining a joint probability distribution over the data lets us generate new data, and also lets us answer all sorts of queries about the data. This is why these models are called *Generative Models*. We will use the Naïve Bayes model trained in previous question to generate data.

(a) **[1pt]** True or false: Given this model's assumptions, any two pixels x_i and x_j where $i \neq j$ are independent given c.

(b) **[1pt]** True or false: Given this model's assumptions, any two pixels x_i and x_j where $i \neq j$ are independent after marginalizing over c.

(c) **[2pts]** Using the parameters fit using MAP in Question 1, produce random image samples from the model. That is, randomly sample and plot 10 binary images from the marginal distribution $p(\mathbf{x}|\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\pi}})$. Hint: To sample from $p(\mathbf{x} \mid \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\pi}})$, first sample random variable c from $p(c \mid \hat{\boldsymbol{\pi}})$ using `np.random.choice`, then depending on the value of c, sample x_j from $p(x_j \mid c, \hat{\theta}_{jc})$ for $j = 1, ..., 784$ using `np.random.binomial(1,..)`. These functions can take matrix probabilities as input, so your solution to this part should be a few lines of code.

(d) **[Optional]** One of the advantages of generative models is that they can handle missing data, or be used to answer different sorts of questions about the model. Derive $p(\mathbf{x}_{bottom}|\mathbf{x}_{top}, \boldsymbol{\theta}, \boldsymbol{\pi})$, the marginal distribution of a single pixel in the bottom half of an image given the top half, conditioned on your fit parameters. Hint: you'll have to marginalize over c.

(e) **[Optional]** For 20 images from the training set, plot the top half the image concatenated with the marginal distribution over each pixel in the bottom half. i.e. the bottom half of the image should use grayscale to represent the marginal probability of each pixel being 1 (darker for values close to 1).