Homework 4

Deadline: Tuesday, March 23, at 11:59pm.

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

- Your solutions to Questions 1, 2, 3 as a PDF file, `hw4_writeup.pdf`.
- Your completed Python code for Question 2, as `q2.py`.

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.

Homeworks are individual work. See the webpage for detailed policies.

1. **[1pt] Marginalization and Log-Sum-Exp.** In this question you will learn how to compute log marginal probabilities in a numerically stable way. Suppose that you have a generative model $p(x, i)$ for labeled data (x, i) where i is a label that can be one of $0, 1, 2, \ldots, k$. Recall that the marginal probability $p(x)$ can be computed using the following formula:

$$p(x) = \sum_{i=0}^{k} p(x, i) \tag{0.1}$$

When x is a high-dimensional data point, it is typical for the marginal $p(x)$ and the joint $p(x, i)$ to be extremely small numbers that cannot be represented in floating point. For this reason, we usually report and compute with *log probabilities*.

If we want to compute $\log(p(x))$ only given access to $\log(p(x, i))$, then we can use what is called a *log-sum-exp*:

$$\log\left(\sum_{i=0}^{k} \exp(a_i)\right) \tag{0.2}$$

where $a_i \in \mathbb{R}$ are real numbers. In our example, if $a_i = \log(p(x, i))$, then expression (0.2) would correspond to the log marginal probability $\log(p(x))$ of x.

Unfortunately, computing log-sum-exp naively can lead to numerical instabilities. The numerical instabilities in log-sum-exp are caused by problems that arise when trying to compute exponentials using floating point numbers. Two things can go wrong:

(a) **Underflow.** If `a[i]` is very small, then `np.exp(a[i])` will evaluate to `0`.

(b) **Overflow.** If `a[i]` is very large, then `np.exp(a[i])` will evaluate to `inf`.

The cause of underflow and overflow is that floating point numbers cannot represent numbers arbitrarily close to 0 nor arbitrarily large numbers.

(a) **[1pt]** We have provided code in `q1.py` with two implementations of log-sum-exp: a naive, numerically unstable implementation and a numerically stable one. Modify the elements of `a` so that `logsumexp_unstable` returns `-inf`, and modify the elements of `b` so that `logsumexp_unstable` returns `inf`. Report the two vectors, `a` and `b`, in your write-up.

[1] https://markus.teach.cs.toronto.edu/csc2515-2021-01

(b) **Optional - not for marks** Prove that our numerically stable implementation is correct by proving that

$$\log\left(\sum_{i=0}^{k}\exp(a_i)\right) = \log\left(\sum_{i=0}^{k}\exp\left(a_i - \max_{j=0}^{k}\{a_j\}\right)\right) + \max_{j=0}^{k}\{a_j\} \tag{0.3}$$

Briefly explain why the numerically stable version is robust to underflow and overflow.

Report your answers to the above questions.

2. **[3pts] Gaussian Discriminant Analysis.** For this question you will build classifiers to label images of handwritten digits. Each image is 8 by 8 pixels and is represented as a vector of dimension 64 by listing all the pixel values in raster scan order. The images are grayscale and the pixel values are between 0 and 1. The labels y are $0, 1, 2, \ldots, 9$ corresponding to which character was written in the image. There are 700 training cases and 400 test cases for each digit; they can be found in `a4digits.zip`.

   A skeleton (`q2.py`) is is provided for each question that you should use to structure your code. Starter code to help you load the data is provided (`data.py`). Note: the `get_digits_by_label` function in `data.py` returns the subset of digits that belong to a given class.

   Using maximum likelihood, fit a set of 10 class-conditional Gaussians with a separate, full covariance matrix for each class. Remember that the conditional multivariate Gaussian probability density is given by,

$$p(\mathbf{x} \mid y = k, \boldsymbol{\mu}, \Sigma_k) = (2\pi)^{-d/2}|\Sigma_k|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right\} \tag{0.4}$$

   You should take $p(y = k) = \dfrac{1}{10}$. You will compute parameters μ_{kj} and Σ_k for $k \in (0...9), j \in (1...64)$. You should implement the covariance computation yourself (i.e. without the aid of 'np.cov'). *Hint: To ensure numerical stability you may have to add a small multiple of the identity to each covariance matrix. For this assignment you should add* $0.01\mathbf{I}$ *to each matrix.*

   (a) **[1pt]** Using the parameters you fit on the training set and Bayes rule, compute the average conditional log-likelihood, i.e. $\frac{1}{N}\sum_{i=1}^{N}\log(p(y^{(i)} \mid \mathbf{x}^{(i)}, \theta))$ on both the train and test set and report it. *Hint: you will want to use the log-sum-exp we discussed in Question 1 to your code.*

   (b) **[1pt]** Select the most likely posterior class for each training and test data point as your prediction, and report your accuracy on the train and test set.

   (c) **[1pt]** Compute the leading eigenvectors (largest eigenvalue) for each class covariance matrix (can use `np.linalg.eig`) and plot them side by side as 8 by 8 images.

   Report your answers to the above questions, and submit your completed Python code for `q2.py`.

3. **[3pts] Categorial Distribution.** In this problem you will consider a Bayesian approach to modelling categorical outcomes. Let's consider fitting the categorical distribution, which is a discrete distribution over K outcomes, which we'll number 1 through K. The probability of each category is explicitly represented with parameter θ_k. For it to be a valid probability

distribution, we clearly need $\theta_k \geq 0$ and $\sum_k \theta_k = 1$. We'll represent each observation $\mathbf{x}$ as a 1-of-K encoding, i.e, a vector where one of the entries is 1 and the rest are 0. Under this model, the probability of an observation can be written in the following form:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{k=1}^{K} \theta_k^{x_k}.$$

Suppose you observe a dataset,

$$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}.$$

Denote the count for outcome k as $N_k = \sum_{i=1}^{n} x_k^{(i)}$. Recall that each data point is in the 1-of-K encoding, i.e., $x_k^{(i)} = 1$ if the ith datapoint represents an outcome k and $x_k^{(i)} = 0$ otherwise. In the previous assignment, you showed that the maximum likelihood estimate for the counts was:

$$\hat{\theta}_k = \frac{N_k}{N}.$$

(a) **[1pt]** For the prior, we'll use the Dirichlet distribution, which is defined over the set of probability vectors (i.e. vectors that are nonnegative and whose entries sum to 1). Its PDF is as follows:

$$p(\boldsymbol{\theta}) \propto \theta_1^{a_1-1} \cdots \theta_K^{a_k-1}.$$

What is the probability distribution of the posterior distribution $p(\boldsymbol{\theta} \,|\, \mathcal{D})$?

(b) **[1pt]** Still assuming the Dirichlet prior distribution, determine the MAP estimate of the parameter vector $\boldsymbol{\theta}$. For this question, you may assume each $a_k > 1$.

(c) **[1pts]** Now, suppose that your friend said that they had a hidden $N + 1$st outcome, $\mathbf{x}^{(N+1)}$, drawn from the same distribution as the previous N outcomes. Your friend does not want to reveal the value of $\mathbf{x}^{(N+1)}$ to you. So, you want to use your Bayesian model to predict what *you* think $\mathbf{x}^{(N+1)}$ is likely to be. The "proper" Bayesian predictor is the so-called *posterior predictive distribution*:

$$p(\mathbf{x}^{(N+1)}|\mathcal{D}) = \int p(\mathbf{x}^{(N+1)}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) \, d\boldsymbol{\theta}$$

What is the probability that the $N+1$ outcome was k, i.e., the probability that $x_k^{(N+1)} = 1$, under your posterior predictive distribution? *Hint: A useful fact is that if* $\boldsymbol{\theta} \sim \text{Dirichlet}(a_1, \ldots, a_K)$*, then*

$$\mathbb{E}[\theta_k] = \frac{a_k}{\sum_{k'} a_{k'}}.$$

Report your answers to the above questions.