CPSC 440/540 Machine Learning (Jan-Apr 2023)
Assignment 3 – due Monday March 27th at **11:59pm**

The assignment instructions are the same as for the previous assignments. If you do the assignment with a partner, please only hand in one copy for the group (using the appropriate Gradescope feature).

1 Gamma-Poission Models [20 points]

Consider counts $y \in \{0, 1, 2, 3, \dots\}$ following a Poisson distribution with rate parameter $\lambda > 0$:

$$y \sim \text{Poisson}(\lambda) \qquad p(y \mid \lambda) = \frac{\lambda^y \exp(-\lambda)}{y!}.$$

We'll assume that λ follows a Gamma distribution (the conjugate prior to the Poisson) with shape parameter $\alpha > 0$ and rate parameter $\beta > 0$:

$$\lambda \sim \text{Gamma}(\alpha, \beta) \qquad p(\lambda \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} \exp(-\beta\lambda) = \frac{1}{Z(\alpha, \beta)} \lambda^{\alpha-1} \exp(-\beta\lambda),$$

where Γ is the gamma function.

Compute the following quantities:

(1.1) [5 points] The posterior distribution,

$$p(\lambda \mid y, \alpha, \beta).$$

Answer: TODO

(1.2) [5 points] The marginal likelihood of y given the hyper-parameters α and β,

$$p(y \mid \alpha, \beta) = \int p(y, \lambda \mid \alpha, \beta) \, \mathrm{d}\lambda.$$

Answer: TODO

(1.3) [5 points] The posterior mean estimate for λ,

$$\mathbb{E}_{\lambda|y,\alpha,\beta}[\lambda] = \int \lambda \, p(\lambda \mid y, \alpha, \beta) \, \mathrm{d}\lambda.$$

Answer: TODO

(1.4) [5 points] The posterior predictive distribution for a new independent observation $\tilde{y}$ given y,

$$p(\tilde{y} \mid y, \alpha, \beta) = \int p(\tilde{y}, \lambda \mid y, \alpha, \beta) d\lambda.$$

Answer: TODO

Hint: You should be able to use the form of the gamma distribution to solve all the integrals that show up in this question. You can use $Z(\alpha, \beta) = \frac{\Gamma(\alpha)}{\beta^\alpha}$ to represent the normalizing constant of the gamma distribution,[1] *and use α^+ and β^+ as the updated parameters of the gamma distribution in the posterior.*

[1] Since continuous PDFs integrate to 1, we have $\int \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} \exp(-\beta\lambda) \, \mathrm{d}\lambda = 1$. Thus we have $\int \lambda^{\alpha-1} \exp(-\beta\lambda) \, \mathrm{d}\lambda = \frac{\Gamma(\alpha)}{\beta^\alpha}$. If we write $p(\lambda \mid \alpha, \beta) \propto \lambda^{\alpha-1} \exp(-\beta\lambda))$, then the normalizing constant is $Z(\alpha, \beta) = \frac{\Gamma(\alpha)}{\beta^\alpha}$.

2 Empirical Bayes for Bernoulli Models [30 points]

If you run `main.py eb-base`, it will load a dataset representing the number of people diagnosed with stomach cancer in a set of different regions, and the number of people at risk in the region. The format of the data is as follows:

- `n[j,0]` is number of positive examples in the training set for group j.
- `n[j,1]` is the total number of examples in the training set for group j.
- `n_test[j,0]` is number of positive examples in the testing set for group j.
- `n_test[j,1]` is the total number of examples in the testing set for group j.

The code first shows the negative log-likelihood (NLL) on the test set if we assume that the probability of getting stomach cancer among each group is 0.5. The NLL under this choice is very high, since the number of positives is fairly low (remember that we want to have a low NLL, corresponding to a high likelihood). The code then computes the MLE for each group, and evaluates the NLL with this choice.

(2.1) [2 points] Why do we get `NaN` for the NLL with the MLE values? What is the actual test likelihood supposed to be in this case?

Answer: TODO

(2.2) [3 points] Laplace smoothing corresponds to MAP estimation with a beta prior that has $\alpha = 2$ and $\beta = 2$. What is the test NLL if we use Laplace smoothing to estimate the parameters?

Answer: TODO

(2.3) [3 points] Instead of using MAP estimation and then computing the NLL, we could use the Bayesian approach of computing the negative logarithm of the posterior predictive probability of the test data. What is the total negative-log-likelihood of one-group-at-a-time posterior predictive probabilities, $\sum_{j=1}^{n_{groups}} -\log p(\texttt{n\_test[j, 0]} \mid \mathbf{X}, \alpha, \beta, \texttt{n\_test[j, 1]})$, using a beta prior with $\alpha = 2$ and $\beta = 2$? Explain why you think this is better/worse than the test NLL under MAP estimation.

Answer: TODO

(2.4) [3 points] Instead of modeling each group independently, consider fitting a single Bernoulli model by pooling the data across all groups. In other words, we ignore the groups and treat all the examples as if they came from a single source. Report the test NLL in this pooled setting when using MLE, MAP, and the posterior predictive. Why are these results more similar than when we use independent Bernoullis for each group?

Answer: TODO

(2.5) [3 points] In the pooled data model, you can compute the logarithm of the marginal likelihood of the hyper-parameters given the data as `betaln(a + n1, b + n0) - betaln(a, b)` (which is the ratio of posterior and prior normalizing constants, converted to the log domain; `betaln` is from `scipy.special`). Here, `n1` is the number of 1s, `n0` is the number of 0s, `a` gives the value of α, and `b` gives the value of β. Can you find the value of α and β that optimize the marginal likelihood? (What are they, and what's the likelihood?)

Hint: It may be helpful to parameterize the beta distribution in terms of $m = \alpha/(\alpha+\beta)$ (the proportion of 1s) and $k = (\alpha + \beta)$ (the "strength" of our belief in this ratio).[2] *Try setting m to the MLE value of θ and varying k.*

Answer: TODO

[2] Instead of writing the beta distribution as $p(\theta \mid \alpha, \beta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}$, write it as $p(\theta \mid m, k) \propto \theta^{km-1}(1-\theta)^{k(1-m)-1}$. The set of distributions you can represent is the same, but the way the parameters change the distribution changes.

(2.6) [5 points] In the textbook where this data came from (Johnson and Albert, "Ordinal Data Modeling"), they suggest using an independent hyper-prior over m and k of the form

$$p(m) \propto m^{.01-1}(1-m)^{9.9-1}, \quad p(k) \propto 1/(1+k)^2.$$

The hyper-prior over m is biased towards low values (since we expect the cancer to be rare) but not particularly strong, while the prior over k is similar to what is called a "Jeffreys prior" over scale variables (which would satisfy a particular definition of being an "uninformative prior").

In the pooled data model, find the value of α and β that optimize the marginal likelihood with this hyper-prior (or equivalently, optimize the log of the marginal likelihood with the log of this prior as the regularizer). Up to one decimal place, what are the optimal values of α and β under this hyper-prior? What values of m and k does this correspond to choosing? Hand in your code for computing the objective function that is being optimized.

Hint: For this question, you can use a "brute force" approach to find α and β, by searching over all values of the from `x.y` *for* `x` *in* $\{0, \ldots, 9\}$ *and* `y` *in* $\{0, \ldots, 9\}$ *(but where at least one of* `x` *and* `y` *must be bigger than 0.*

Answer: TODO

(2.7) [5 points] Now that we have a better way to estimate, we can return to the original setting with a separate parameter for each group. The marginal likelihood is then given by just the product of what it was for the pooled data:

$$p(\mathbf{X} \mid \alpha, \beta) \propto \prod_{j=1}^{k} \frac{Z(\alpha + n_{1j}, \beta + n_{0j})}{Z(\alpha, \beta)},$$

where n_{1j} is the number of 1s in group j, n_{0j} is the number of 0s in group j, and the normalizing constant Z is the beta function.

Find the values of α and β that optimize the marginal likelihood times the hyper-prior from the previous question. You'll need to change the limits on the brute-force search; keep in mind the meanings of α and β and that cancer is (thankfully) relatively rare when you're deciding on the new limits. Hand in your code to compute the objective function being optimized, report the values of α and β that you find, and report the negative logarithm of the posterior predictive probability of the test data under this choice of α and β. If this is too slow, you can restrict the search to positive integer values for α and β.

Answer: TODO

(2.8) [2 points] In the model from the last question, what is the posterior predictive probability of seeing a 1 for a new group?

Answer: TODO

(2.9) [2 points] Under the choice of α and β from part (2.7), what is the NLL of the test data if we use MAP estimation for the parameters?

Answer: TODO

(2.10) [2 points] In this question, many of the best-performing methods achieve similar performance. Give an explanation for why these different model choices do not make a big difference for this dataset.

Answer: TODO

3 More Deep Learning [24 points]

The code for this question uses PyTorch; see `https://pytorch.org` *for installation instructions (the CPU version is fine; if you're having trouble, you could also use Google Colab). This tutorial page is a decent starting place for reference.*

`code/neural_net.py` has our hand-coded neural net from the last assignment, and also an autodiff-based one using PyTorch's neural net interface. The two should be the same as written here; `main.py nns-35` runs the same 3-vs-5 classification problem as last time for both models. Take a look, and make sure things more or less make sense.

(3.1) [3 points] On the previous assignment, we concatenated all parameters into one big vector of parameters w. What's an advantage of not doing that, and just keeping separate weight matrices for each layer?

Answer: TODO

(3.2) [3 points] For the provided code in `TorchNeuralNetClassifier(layer_sizes=[3])`, how many parameters are there, and what do they represent?

Hint: `list(thing.parameters())` *will get you the parameters of either a layer or a* `torch.nn.Module`*.*

Answer: TODO

(3.3) [4 points] Modify the network to use a more typical classification loss: maximizing the likelihood of a categorical likelihood, aka softmax loss, aka cross-entropy. Hand in the modifications to your code.

You don't need to implement it yourself from scratch – you can use anything from PyTorch here.

Answer: TODO

(3.4) [3 points] PyTorch doesn't have an easy built-in way to compute the loss after going through a `torch.nn.Softmax` layer. Why not? Why does it insist on log inputs? Describe an example where using log-probabilities would give meaningfully different results from "plain" probabilities.

Answer: TODO

(3.5) [4 points] Change `TorchNeuralNetClassifier` to work well on the 10-way actual MNIST dataset, which is run in `main.py nns-10way`. You'll probably do similar stuff to what you did last time, but use the softmax loss *(if you want, also play around with L2 loss to compare)*. Describe the changes you made and your best test performance.

Answer: TODO

(3.6) [4 points] Change the model in `Convnet.build` to use convolutional layers, while getting test error at least as good as the MLP got. (Feel free to make it less generic, e.g. not supporting arbitrary `layer_sizes` anymore.) Submit your `build()` method, and any other relevant changes, as well as your final test error. (We're probably overfitting a bit at this point, though. . . .)

Hint: To make it a little faster by using your GPU, you can pass `device="cuda"` *if you have an appropriate version of PyTorch installed, or* `device="mps"` *if you're a recent Mac.*

(3.7) [3 points] If we use a layer `torch.nn.Conv2d(in_channels=1, out_channels=128, kernel_size=(4, 4))`, what are the resulting parameter shapes, and why? Why don't we need to pass in the shape of the input image?

Answer: TODO

4 Very-Short Answer Questions [26 points]

Each of these is worth [2 points].

(4.1) When we use categorical variables, we assume that the categories do not have any special ordering. But we when sample from a categorical distribution, we use the CDF $p(x \leq c)$. Why does this make sense – isn't it "enforcing an order"?

Answer: TODO

(4.2) Monte Carlo methods approximate the expectation of a function of a random variable, but we said that they can be used to approximate probabilities. What random function would you use to approximate the probability of an event?

Answer: TODO

(4.3) How do MAP and Bayesian methods differ in how they make predictions?

Answer: TODO

(4.4) Why do we say that Bayesian methods incorporate regularization, even though there is no regularization term in the posterior predictive distribution?

Answer: TODO

(4.5) How does using a conjugate prior simplify the marginal likelihood calculation?

Answer: TODO

(4.6) What is a hyper-hyper-parameter?

Answer: TODO

(4.7) Suppose we use the tabular parameterization for multi-class classification with d features, where we have k classes and each feature can take k values. What is the exact number of parameters we need, if we remove redundant parameters by exploiting that probabilities sum to 1?

Answer: TODO

(4.8) In the softmax function we have k weight vectors, one for each class. But in the special case of the sigmoid function we fix one of these weight vectors to zero, so we only have one weight vector for two classes. If we instead treat binary classification as a two-class problem with $k = 2$ weight vectors, would the set of classification functions our model can represent be different? Would our learning algorithm find the same function?

Answer: TODO

(4.9) If we use a neural network with one hidden layer for multi-class classification, what are the sizes of the parameter matrices W and V, and what is the computational cost of backpropagation for one example? You can use k as the number of classes and m as the number of hidden units.

Answer: TODO

(4.10) Why can RNNs label sequence of different lengths?

Answer: TODO

(4.11) When predicting with sequence-to-sequence RNNs we do not know the length of the output sequence. How is the length of the output sequence determined during decoding?

Answer: TODO

(4.12) Why do we use "context vectors" instead of including all encoding states in attention models?

Answer: TODO

(4.13) Why do we include "position encodings" in Trasformers?

Answer: TODO