# CPSC 440/550 Advanced Machine Learning (Jan-Apr 2024)
# Assignment 3 – due Tuesday April 2nd 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 [25 points]

The Poisson distribution is a discrete distribution over $\{0, 1, 2, 3, \dots\}$. Given a "rate" parameter $\lambda > 0$, the distribution is given by

$$y \sim \text{Poisson}(\lambda) \qquad p(y \mid \lambda) = \frac{1}{y!}\lambda^y e^{-\lambda} \qquad \mathbb{E}[y \mid \lambda] = \lambda \qquad \text{Var}[y \mid \lambda] = \lambda.$$

Using $\lambda^y = \exp(\log\lambda \cdot y)$, the Poisson can be written as a one-parameter exponential family as

$$p(y \mid \lambda) = \frac{h(y)}{Z_1(\lambda)}\exp(\eta(\lambda)s(y)) \qquad \underbrace{\eta(\lambda) = \log\lambda}_{\text{parameter function}} \qquad \underbrace{s(y) = y}_{\text{suff. stat.}} \qquad \underbrace{h(y) = \frac{1}{y!}}_{\text{support func.}} \qquad \underbrace{Z_1(\lambda) = e^{\lambda}}_{\text{normalizer}}.$$

Or, in canonical form with the log-rate parameter $\eta = \log\lambda$,

$$p(y \mid \eta) = \frac{h(y)}{Z_2(\eta)}\exp(\eta\, s(y)) \qquad \underbrace{s(y) = y}_{\text{suff. stat.}} \qquad \underbrace{h(y) = \frac{1}{y!}}_{\text{support func.}} \qquad \underbrace{Z_2(\eta) = e^{e^{\eta}}}_{\text{normalizer}}.$$

The usual conjugate prior for the Poisson distribution is the Gamma, which can be written with a shape parameter $\alpha > 0$ and a rate parameter $\beta > 0$ as

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

where $\Gamma$ is the gamma function, so that e.g. for positive integers $n$ we have $\Gamma(n) = (n-1)!$, and $Z_g(\alpha, \beta) = \Gamma(\alpha)/\beta^\alpha$. This is itself a two-parameter exponential family (albeit not written here in canonical form).

Let's define a standard notation $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y^{(i)}$; you may or may not find this useful in your answers.

*Hint: You should be able to use the form of the gamma distribution to solve all the integrals that show up in this question. Use $Z_g$ to represent the normalizing constant of the gamma distribution,*[1] *and write e.g. $\alpha^+$ and $\beta^+$ as the updated parameters of the gamma distribution in the posterior.*

**[1.1]** [5 points] Calculate the posterior distribution for $\lambda$, $p(\lambda \mid \mathbf{y}, \alpha, \beta)$.

Answer: TODO

**[1.2]** [5 points] Calculate the marginal likelihood of $\mathbf{y}$ given the hyper-parameters $\alpha$ and $\beta$,

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

Answer: TODO

[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 $Z(\alpha, \beta) = \frac{\Gamma(\alpha)}{\beta^\alpha}$.

**[1.3]** [5 points] Calculate the posterior mean estimate for $\lambda$,

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

Answer: TODO

**[1.4]** [5 points] Calculate the posterior predictive distribution for a new independent observation $\tilde{y}$ given $\mathbf{y}$,

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

Answer: TODO

**[1.5]** [5 points] Poisson regression is a standard method for predicting a count $y$ from an input $x$, based on a dataset $\left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$. It parameterizes $y \mid x \sim \operatorname{Poisson}(\exp(w^\mathsf{T} x))$, i.e. setting the natural parameter $\eta$ to the output of a linear model, which also implies that $\log \mathbb{E}[y \mid x] = w^\mathsf{T} x$. Probably the most common use of this method is frequentist, using the MLE for $w$.

Suppose the counts $y$ in our dataset are mostly relatively small, and number of data points $n$ is also small; thus we would like to incorporate Bayesian uncertainty into our model. Using the results from this question and our discussion from class of Bayesian linear/logistic regression, what would be a reasonable way to do that? Specify a reasonable probabilistic model, and describe in general terms how to use it.

Answer: TODO

## 2 Empirical Bayes for Bernoulli Models [40 points]

If you run `main.py eb-base`, it will load a small 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]** [3 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]** [4 points] Laplace smoothing corresponds to MAP estimation with a beta prior that has $\alpha = 2$ and $\beta = 2$. There's a stub in `main.py` under `eb_map`. What is the test NLL if we use Laplace smoothing to estimate the parameters?

Answer: TODO

**[2.3]** [4 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]** [4 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]** [4 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 $\alpha$, and `b` gives the value of $\beta$. Can you find the value of $\alpha$ and $\beta$ 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 $\theta$ and varying $k$.*

[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.

Answer: TODO

**[2.6]** [6 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 $\alpha$ and $\beta$ 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 $\alpha$ and $\beta$ 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 $\alpha$ and $\beta$, by searching over $\{0.1, 0.2, \ldots, 9.9\}$ for each value.*

Answer: TODO

**[2.7]** [6 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 $\alpha$ and $\beta$ 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 $\alpha$ and $\beta$ 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 $\alpha$ and $\beta$ that you find, and report the negative logarithm of the posterior predictive probability of the test data under this choice of $\alpha$ and $\beta$. If this is too slow, you can restrict the search to positive integer values for $\alpha$ and $\beta$.

Answer: TODO

**[2.8]** [3 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]** [3 points] Under the choice of $\alpha$ and $\beta$ from Question [2.7], what is the NLL of the test data if we use MAP estimation for the parameters?

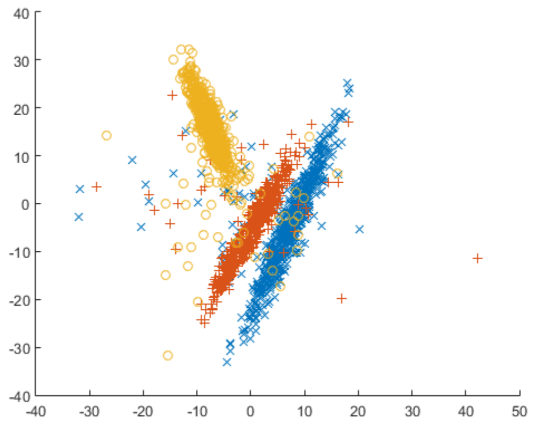
Answer: TODO

**[2.10]** [3 points] In this question, many of the best-performing methods achieve fairly-similar performance. Give an explanation for why these different model choices do not make a *huge* difference for this dataset.

Answer: TODO

## 3 Generative Classifiers with Gaussian Assumption [35 points]

Consider the 3-class classification dataset in this image:



In this dataset, we have 2 features and each colour represents one of the classes. Note that the classes are highly structured: the colours each roughly follow a Gausian distribution plus some noisy samples.

Since we have an idea of what the features look like for each class, we might consider classifying inputs $x$ using a generative classifier. In particular, we are going to use Bayes rule to write

$$p(y = c \mid x, \Theta) = \frac{p(x \mid y = c, \Theta) \cdot p(y = c \mid \Theta)}{p(x \mid \Theta)},$$

where $\Theta$ represents the parameters of our model. To classify a new example $\tilde{x}$, we get

$$\hat{y} = \operatorname*{arg\,max}_{c \in \{1,2,\dots,k\}} p(y = c \mid \tilde{x}, \Theta) = \operatorname*{arg\,max}_{c \in \{1,2,\dots,k\}} p(y = c \mid \Theta) p(\tilde{x} \mid y = c, \Theta) = \operatorname*{arg\,max}_{c \in \{1,2,\dots,k\}} \pi_c \mathcal{N}(\tilde{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c),$$

where we've assumed that $y \mid \Theta \sim \text{Categorical}(\pi_1, \dots, \pi_k)$, and $x \sim (y = c) \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$; the notation $\mathcal{N}(x \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ means the density of $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ evaluated at $x$.

Recall that the maximum likelihood estimate for $\pi_c$ is simply $n_c/n$, where $n_c = \sum_{i=1}^n \mathbb{1}(y^{(i)} = c)$ is the number of data points in class $c$. Given a dataset $(\mathbf{X}, \mathbf{y})$, the MLE for the mean and covarance parameters is then

$$\begin{aligned}
&\operatorname*{arg\,max}_{\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_1,\dots,\boldsymbol{\Sigma}_k} \prod_{i=1}^{n} \pi_{y^{(i)}} \mathcal{N}(x^{(i)} \mid \boldsymbol{\mu}_{y^{(i)}}, \boldsymbol{\Sigma}_{y^{(i)}}) \\
&\qquad = \operatorname*{arg\,min} \sum_{i=1}^{n} \left( \log \left| \boldsymbol{\Sigma}_{y^{(i)}} \right| + \left( x^{(i)} - \boldsymbol{\mu}_{y^{(i)}} \right)^{\mathsf{T}} \boldsymbol{\Sigma}_{y^{(i)}}^{-1} \left( x^{(i)} - \boldsymbol{\mu}_{y^{(i)}} \right) \right) \\
&\qquad = \operatorname*{arg\,min} \sum_{c=1}^{k} \sum_{i : y^{(i)} = c} \left( \log |\boldsymbol{\Sigma}_c| + \left( x^{(i)} - \boldsymbol{\mu}_c \right)^{\mathsf{T}} \boldsymbol{\Sigma}_c^{-1} \left( x^{(i)} - \boldsymbol{\mu}_c \right) \right).
\end{aligned}$$

In class, we discussed the general GDA/QDA model, which allows any value for the $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$. The objective above then separates for the different $c$, and the MLE becomes just the per-class MLE.

We also discussed *linear discriminant analysis* (LDA), which enforces that all the covariance matrices agree, $\boldsymbol{\Sigma}_c = \boldsymbol{\Sigma}$. In this case the classifier becomes linear, with $\boldsymbol{\Sigma}$ corresponding to a "pooled covariance."

Another common restriction on the $\mathbf{\Sigma}_c$ is that they are diagonal matrices, since this only requires $\mathcal{O}(d)$ parameters instead of $\mathcal{O}(d^2)$; this corresponds to assuming that the features are independent univariate Gaussians given the class label.

**[3.1]** [10 points] Derive the MLE for the GDA model under the assumption of *common diagonal covariance* matrices, $\mathbf{\Sigma}_c = \operatorname{diag}(\sigma_1^2, \dots, \sigma_d^2)$. Note that this means there are $d$ total parameters for the model's covariances (plus $kd$ parameters in each class's separate mean, $\boldsymbol{\mu}_c$).

*Hint: You might be able to significantly simply notation if you write something like $\sum_{i:y^{(i)}=c}$, or define some shortcut notation like $\sum_{i \in y_c}$ to mean the same thing; just make sure it's clear. You also might want to use $n_c = \sum_{i \in y_c} 1$ to be the number of cases where $y^{(i)} = c$.*

*Hint: The determinant of a diagonal matrix is the product of its diagonal entries. The inverse is the diagonal matrix where each diagonal entry is inverted.*

Answer: TODO

**[3.2]** [10 points] Derive the MLE for the GDA model under the assumption of *individual scaled-identity* matrices, $\mathbf{\Sigma}_c = \sigma_c^2 \mathbf{I}$; here there are $k$ total covariance parameters, one per class.

Answer: TODO

**[3.3]** [10 points] When you run `main.py gda` it loads a variant of the dataset in the figure that has 12 features and 10 classes. This data has been split up into a training and test set, and the code fits a $k$-nearest neighbour classifier to the training set then reports the accuracy on the test data (around $\sim 63\%$ test error). The $k$-nearest neighbour model does poorly here since it doesn't take into account the Gaussian-like structure in feature space for each class label.

Fill in the class `GDA` to fit a GDA model to this dataset, using the MLE with individual *full* covariance matrices. Hand in your code, and report the test set accuracy.

*Hint: You can use the result from class about the MLE for this model. You can also feel free to use `scipy.stats.multivariate_normal`, or code the relevant bits yourself, as you prefer.*

*Hint: You probably want to handle everything "in log space" for your computation.*

Answer: TODO

**[3.4]** [5 points] The data here has some outliers. Let's try to replace the Gaussian distribution of the previous solution with a more robust multivariate $t$ distribution. Unfortunately, this distribution doesn't have a closed-form MLE, but `student_t.MultivariateT` implements numerical optimization using PyTorch. Fill in the class `generative_classifiers.TDA`, called by `main.py tda`, to fit a generative model based on the multivariate $t$ distribution. Report your test accuracy and hand in your code.

*Hint: These take a little longer to fit; you might want to print some kind of status update as you go to make sure it's happening. The `tqdm` package is a nice, easy version: `from tqdm import tqdm; for c in tqdm(range(k)): ...`*

Answer: TODO