

CPSC 440/550 Advanced Machine Learning (Jan-Apr 2025)

Assignment 2 – due Sunday March 22 at 11:59pm

The assignment instructions are the same as for the previous assignment, but for this assignment you can work in groups of 1 or 2. Please only hand in one assignment for the group, and use the Gradescope group feature.

1 Categorical Inference [20 points]

Consider a categorical distribution that takes value 1 with probability 0.42, 2 with probability 0.24, and 3 with probability 0.34.

- [1.1] [2 points] Suppose we generate four iid samples $\{x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}\}$ according to this model. What is the probability that all four samples are equal to 3?

Answer: TODO

- [1.2] [2 points] Write a Python function that generates t iid examples from this distribution, based on numpy's `rng.random()` function (which returns uniformly pseudo-random numbers in $[0, 1]$) as the source of randomness. Do not use other random generation functions or functions designed specifically for this purpose, though “general-purpose” numpy routines are fine. Your implementation does not need to be particularly efficient. Hand in your code.

Answer: TODO

- [1.3] [2 points] Consider a “game” where you get \$15 if a sample from this distribution is even, and lose \$4 if the sample is odd. What is the expected \$ value you get from playing this game once?

Answer: TODO

- [1.4] [6 points] If you weren't able to do that math, one way to approximate the answer of the previous question is to take

$$\frac{1}{t} \sum_{i=1}^t f(x^{(i)}),$$

where each $x^{(i)}$ is an iid sample, and f gives the \$ value for that sample (either 15 or -4).

Implement this approximation, and use the code in `expval_mc` (inside `main.py`) to plot the following: draw a line of the “running mean” of the approximation as you see more samples, going from 1 to 100,000 samples. Show three independent runs on the same plot, in different colours (just calling `ax.plot` more than once and using its default colour cycle is fine). Also use `ax.axhline` to show the analytical expected value from the previous part. Hand in this plot.

Answer: TODO

- [1.5] [8 points] Now, suppose that you start playing the game with \$40. You'd really like to turn that into \$300 to have an exorbitant night on the town, but if you ever can't pay your debt (that is, your balance goes below \$0) then you need to go on the lam. What's the probability that you'll be able to have a fun night? Answer to within 1%, i.e. if the answer were exactly 21.34% (it's not) then it's okay if your answer is anywhere between 20.34% and 22.34%. Hand in your code and a justification that your answer is accurate enough.

You could use Monte Carlo, or you could compute it analytically, or various other approaches. Remember not to look up specific homework problems online!

Answer: TODO

2 Maximizing a simple expectation [25 points]

- [2.1] [5 points] When we derived the EM algorithm for a general $p(\mathbf{X}, \mathbf{Z} \mid \theta)$ with a variational distribution q (depending on $\mathbf{X}$ and θ) over the variable $\mathbf{Z}$, we used

$$\text{ELBO}_q(\mathbf{X}) = \mathbb{E}_{\mathbf{Z} \sim q} \log p(\mathbf{X}, \mathbf{Z} \mid \theta) + \text{Entropy}(q).$$

When deriving VAEs, we instead used a per-sample ELBO with a variational distribution q specifically for the Z corresponding to a given X :

$$\text{ELBO}_{q_x}(x) = \mathbb{E}_{Z \sim q_x} \log p(x, Z \mid \theta) + \text{Entropy}(q_x).$$

Show that if $(x^{(i)}, z^{(i)})$ are iid under p and $q(\mathbf{Z}) = \prod_{i=1}^n q_i(z^{(i)})$, then $\text{ELBO}_q(\mathbf{X}) = \sum_{i=1}^n \text{ELBO}_{q_i}(x^{(i)})$. You may assume that z is discrete, if you prefer, although it won't fundamentally change the proof.

Answer: TODO

Now, let's consider a particular problem: the one-dimensional data distribution

$$Z \sim \text{Bern}(1/2) \quad X \mid (Z = z) \sim \mathcal{N}(\mu z, 1)$$

so that the marginal distribution of X is simply $\frac{1}{2}\mathcal{N}(0, 1) + \frac{1}{2}\mathcal{N}(\mu, 1)$. Given n iid samples $\mathbf{X} = (x^{(1)}, \dots, x^{(n)})$ from X , we'd like to estimate the single parameter μ .

Here $z^{(i)}$ is a binary random variable; we can therefore say that $q_i = \text{Bern}(r^{(i)})$, for a parameter $r^{(i)} \in [0, 1]$. Let $\psi(t) = -t \log t - (1-t) \log(1-t)$ be a function defined on $(0, 1)$.

- [2.2] [5 points] Show that, given parameters μ and $r^{(i)}$, we have in this problem that

$$\text{ELBO}_{q_i}(x^{(i)}) = -\frac{1}{2} \log(8\pi) - \frac{1}{2} \left(x^{(i)}\right)^2 + \mu r^{(i)} x^{(i)} - \frac{1}{2} \mu^2 r^{(i)} + \psi(r^{(i)}). \quad (2.1)$$

Answer: TODO

- [2.3] [4 points] In the argument deriving the E step of EM, we appealed to the KL divergence to show that maximizing the ELBO matches q_i to the conditional distribution of $Z^{(i)} \mid x^{(i)}$. In this case, that's the Bernoulli with parameter equal to the responsibility

$$\begin{aligned} r^{(i)} &= p(Z^{(i)} = 1 \mid \theta, x^{(i)}) \\ &= \frac{p(Z^{(i)} = 1)p(x^{(i)} \mid Z^{(i)} = 1, \theta)}{p(Z^{(i)} = 0)p(x^{(i)} \mid Z^{(i)} = 0, \theta) + p(Z^{(i)} = 1)p(x^{(i)} \mid Z^{(i)} = 1, \theta)} \\ &= \frac{\frac{1}{2}p(x^{(i)} \mid Z^{(i)} = 1, \theta)}{\frac{1}{2}p(x^{(i)} \mid Z^{(i)} = 0, \theta) + \frac{1}{2}p(x^{(i)} \mid Z^{(i)} = 1, \theta)} \\ &= \frac{e^{-\frac{1}{2}(x^{(i)} - \mu)^2}}{e^{-\frac{1}{2}(x^{(i)})^2} + e^{-\frac{1}{2}(x^{(i)} - \mu)^2}} \\ &= \frac{1}{e^{-\frac{1}{2}(x^{(i)})^2} e^{\frac{1}{2}(x^{(i)} - \mu)^2} + 1} \\ &= \frac{1}{1 + e^{-\mu x^{(i)} + \frac{1}{2}\mu^2}} = \sigma\left(\mu x^{(i)} - \frac{1}{2}\mu^2\right), \end{aligned}$$

where $\sigma(t) = 1/(1 + \exp(-t))$ is the logistic sigmoid function. Checking that this result makes sense: as $\mu x^{(i)} \rightarrow \infty$, $r^{(i)} \rightarrow 1$; as $\mu x^{(i)} \rightarrow -\infty$, $r^{(i)} \rightarrow 0$; if $x^{(i)} = \mu/2$, then $r^{(i)} = \frac{1}{2}$.

Show directly from (2.1), without appealing to the responsibility or the KL divergence argument, that this is indeed the same value which maximizes $\text{ELBO}_{q_i}(x^{(i)})$ for a given μ .

Answer: TODO

[2.4] [4 points] Show that, for fixed $r^{(1)}, \dots, r^{(n)}$, the $\text{ELBO}_q(\mathbf{X})$ is maximized when $\mu = \frac{\sum_i r^{(i)} x^{(i)}}{\sum_i r^{(i)}}$.

Answer: **TODO**

[2.5] [7 points] The function `plot_em` in `main.py` loads some data sampled from this distribution, with some particular value for μ . Fill in the code to compute the `log_liks` array and to run EM; [hand in your code and the resulting figure](#), which should show EM quickly converging towards the maximum of this 1d likelihood function.

Answer: **TODO**

3 Naive Bayes [25 points]

Let's fit the following naive Bayes model on binarized MNIST:

$$Y \sim \text{Cat}(\theta) \quad \theta_j \sim \text{Beta}(\alpha, \beta) \quad X_j \mid (Y = y, \theta_j) \sim \text{Bern}(\theta_{j|y}).$$

(We don't really need to put a prior on the Y parameter θ , since we have plenty of samples there, but if we don't use a prior on θ_j we'll get some NaNs.)

[3.1] [10 points] Implement MAP in this model in `naive_bayes.py`. Hand in your code, and the output of `python main.py mnist-nb`, which tests with several different choices for $\alpha = \beta$.

Answer: TODO

This model is okay, but not great. How much better can we expect to get? `python main.py mnist-logreg` will fit a multiclass logistic regression (aka softmax regression) model from scikit-learn, which gets about 7.6% error, less than half the error rate of naive Bayes. (Various nonlinear models can get almost zero error on this problem.) So, let's try a way to make naive Bayes less naive, which will substantially improve its performance.

It seems reasonable that there would be clusters in the classes. For example, there might be several different "general ways" to draw the digit 7. Instead of assuming that the features are independent given the class label, we might instead assume they're independent given the *cluster* that they are in.

We could treat this all probabilistically, but that'd entail a somewhat complicated EM training loop. Instead, let's consider a *vector-quantized naive Bayes* (VQNB) implementing a version of this idea:

- It clusters the examples associated with *each digit* into k clusters, using k-means clustering (k clusters for each class, $10k$ clusters total). We'll use $z^{(i)}$ as the cluster number of example i . The value $z^{(i)}$ will be from 1 to k , with $y^{(i)}$ determining whether we are considering the k "0" clusters, the k "1" clusters, or so on.
- Since we don't know $z^{(i)}$ (it is a "latent variable"), we can marginalize over its possible values. Using the marginalization and then the product rule, we can write the joint probability as

$$\begin{aligned} p(x_1^{(i)}, x_2^{(i)}, \dots, x_d^{(i)}, y^{(i)}) &= \sum_{z=1}^k p(x_1^{(i)}, x_2^{(i)}, \dots, x_d^{(i)}, y^{(i)}, z^{(i)} = z) \\ &= \sum_{z=1}^k p(x_1^{(i)}, x_2^{(i)}, \dots, x_d^{(i)} \mid y^{(i)}, z^{(i)} = z) p(y^{(i)}, z^{(i)} = z) \\ &= \sum_{z=1}^k p(x_1^{(i)}, x_2^{(i)}, \dots, x_d^{(i)} \mid y^{(i)}, z^{(i)} = z) p(z^{(i)} = z \mid y^{(i)}) p(y^{(i)}) \\ &= p(y^{(i)}) \sum_{z=1}^k p(z \mid y^{(i)}) \left[\prod_{j=1}^d p(x_j^{(i)} \mid y^{(i)}, z) \right]; \end{aligned}$$

the last line uses independence of the features given the cluster.

[3.2] [12 points] Implement the VQNB method (there's a stub in `naive_bayes.py`). Hand in your code, and the test error you obtain with this model for $k = 2$ through $k = 5$.

Hint: The same `KMeans` class as last time is in the handout code for you to use, or feel free to use `scikit-learn`'s.

Hint: $p(y)$ you can handle just as before. $p(z \mid y)$ is a categorical variable. $p(x_j \mid y, z)$ is Bernoulli; you'll have one Bernoulli parameter for each (y, z) pair, which it might be convenient to organize in a $10 \times k$ array.

Hint: If you're working with log-probabilities (which is a good idea), `scipy.special.logsumexp` might be helpful for the sum operation.

Hint: To help with debugging, note that you should get the naive Bayes model in the special case of $k = 1$. Further, with this type of model you usually see the biggest performance gain when going from $k = 1$ to $k = 2$.

Hint: You probably won't be able to exactly match the performance of logistic regression, but it'll get closer.

Answer: **TODO**

- [3.3] [3 points] For a run of the method with $k = 5$, show the images obtained by plotting the estimates for $p(x_j = 1 \mid z, y)$ for all j as a 28 by 28 image, for each value of z and y (so there should be 50 images). There's a code stub in there for plotting it, just fill that out slightly.

Answer: **TODO**

4 Generative Gaussians Galore [30 points]

The following is the probabilistic model for linear discriminant analysis (LDA):

$$Y \sim \text{Cat}(\pi) \quad (X \mid Y = c) \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}).$$

LDA is the generative classifier $\arg \max_c p(Y = c \mid X = x)$.

If you get an import error saying something about sklearn, pip/conda/otherwise install scikit-learn.

- [4.1] [7 points] There is a partial implementation of LDA in `lda.py`. Finish the `fit()` function to implement the MLE, and [submit your code and the figure generated by `main.py plot-boundaries`](#), which shows the decision boundaries for both your LDA and scikit-learn’s multinomial logistic regression.

While you probably won’t see an obvious difference between a “naive” implementation and one that does linear algebra “nicely” on this small dataset, you’ll get [2 bonus points] if your implementation uses “good” linear algebra; here the best thing to do is probably to use the Cholesky decomposition (`scipy.linalg.solve_cho` or `scipy.linalg.solve_triangular`).

Answer: **TODO**

- [4.2] [4 points] `main.py learning-curves` produces training and test accuracy curves as a function of training set size n for both logistic regression and your LDA implementation. (The lines show median accuracies, and areas around them interquartile ranges, the spread from the 25th percentile to 75th.) It does this both for the 2d data distribution we were just looking at, and then for a 17-dimensional distribution that’s the same but adds independent noise dimensions: $X_3 \sim \mathcal{N}(0, 1)$, $X_4 \sim \mathcal{N}(0, 2)$, $\dots$, $X_{17} \sim \mathcal{N}(0, 15)$. [Include both plots, and explain the relevant patterns you see.](#)

You might get some numerical errors for the lower sample sizes in the higher-dimensional part. You can avoid this by adding a very small amount, say $1e-8$, to the diagonal of your covariance estimate. This is technically not quite the MLE anymore, but it’s fine.

Answer: **TODO**

Now, we’ll turn to another use of LDA that we didn’t describe in class: as a form of supervised dimensionality reduction.

- [4.3] [4 points] Let the global mean of the distribution, $\sum_c \pi_c \boldsymbol{\mu}_c$, be $\boldsymbol{\mu}$, and define $\mathbf{m}_c = \boldsymbol{\mu}_c - \boldsymbol{\mu}$. Let $f_c(x) = \log p(y = c \mid x) + \text{const}$ so that $\arg \max_c f_c(x)$ gives the optimal classifier; [write \$f_c\$ as a function only of \$\(x - \boldsymbol{\mu}\)^\top \boldsymbol{\Sigma}^{-1} \mathbf{m}_c\$, plus something depending on \$\mathbf{m}_c\$ and \$\pi_c\$ \(but not \$x\$ \).](#)

Answer: **TODO**

Since this is true for each c , this means that the optimal classification function “cares” only about x in the dot products of $x - \boldsymbol{\mu}$ with each of the $\boldsymbol{\Sigma}^{-1} \mathbf{m}_c$: that is, only the part of $x - \boldsymbol{\mu}$ in the subspace $\mathcal{M} = \text{span}(\{\boldsymbol{\Sigma}^{-1} \mathbf{m}_1, \dots, \boldsymbol{\Sigma}^{-1} \mathbf{m}_k\})$ matters, and any component of $x - \boldsymbol{\mu}$ outside of that is just irrelevant. Since we know that $\sum_c \pi_c \mathbf{m}_c = \sum_c \pi_c (\boldsymbol{\mu}_c - \sum_{c'} \pi_{c'} \boldsymbol{\mu}_{c'}) = 0$, we also have $\sum_c \pi_c \boldsymbol{\Sigma}^{-1} \mathbf{m}_c = 0$, and hence $\mathcal{M}$ is of dimension at most $k - 1$. So, in this probabilistic model, there are at most $k - 1$ directions that matter: everything else is just noise. We also know exactly how to estimate these directions: just estimate $\boldsymbol{\Sigma}$ and each of the $\boldsymbol{\mu}_c$, then plug in.

Our last step: what if I don’t want $k - 1$ dimensions? There’s no point in going to something higher, but what if I want to go to fewer than $k - 1$ dimensions? Let’s start by asking what we should do to go to *one* dimension.

Let's try looking at a linear mapping $w^\top X$ for a random X . We can rewrite $X = \mathbf{L}Z + \boldsymbol{\mu}_Y$, where Y is the categorical label, Z is standard normal, and $\mathbf{L}$ is the same shape as $\boldsymbol{\Sigma}$ such that $\mathbf{L}\mathbf{L}^\top = \boldsymbol{\Sigma}$. Then

$$w^\top X = w^\top \mathbf{L}Z + w^\top \boldsymbol{\mu}_Y.$$

The first term here is classification-irrelevant noise, $w^\top \mathbf{L}Z \sim \mathcal{N}(0, w^\top \boldsymbol{\Sigma} w)$. The second term, $w^\top \boldsymbol{\mu}_Y$, is the signal that actually tells us anything about Y (and, clearly, depends *only* on Y). What's its variance?

[4.4] [3 points] Show that $\text{Var}(w^\top \boldsymbol{\mu}_Y) = w^\top \boldsymbol{\Sigma}_B w$, where $\boldsymbol{\Sigma}_B = \sum_c \pi_c (\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^\top$ is the between-group (co)variance.

Hint: This is basically a one-line question about the definition of the covariance matrix.

Answer: TODO

For our projection to be informative, we want to choose a w such that $w^\top \boldsymbol{\Sigma}_B w$ is as big as possible relative to $w^\top \boldsymbol{\Sigma} w$.

The following standard fact about the “Rayleigh quotient” may be useful.

Lemma 4.1. For a symmetric matrix A with eigenvalues λ_i and corresponding orthonormal eigenvectors q_i , $\max_{w \neq 0} \frac{w^\top A w}{\|w\|^2} = \max_i \lambda_i$, with the maximum achieved by any w in $\text{span}(\{q_i : \lambda_i = \max_j \lambda_j\})$.

Proof. Write w in the eigenbasis of A : $w = \sum_i a_i q_i$. Then $w^\top w = \sum_i a_i^2$, and since $A = \sum_i \lambda_i q_i q_i^\top$, we similarly have $w^\top A w = \sum_i a_i^2 \lambda_i$. Thus the objective function can be written $\sum_i \frac{a_i^2}{\sum_j a_j^2} \lambda_i$, a weighted mean of the eigenvalues. This is maximized iff all nonzero a_i are on the maximal λ_i . $\square$

[4.5] [5 points] Assuming that $\boldsymbol{\Sigma}$ is strictly positive definite, show that

$$\arg \max_{w \neq 0} \frac{w^\top \boldsymbol{\Sigma}_B w}{w^\top \boldsymbol{\Sigma} w}$$

can be given by finding a top eigenvector v (an eigenvector corresponding to the largest eigenvalue $\lambda_{\max}$) of the positive-definite matrix $\mathbf{L}^{-1} \boldsymbol{\Sigma}_B \mathbf{L}^{-\top}$, then taking $w = \mathbf{L}^{-\top} v$.

Also show that this implies $\boldsymbol{\Sigma}_B w = \lambda_{\max} \boldsymbol{\Sigma} w$, known as a “generalized eigenvector.”

Answer: TODO

Extending the logic of the previous part, we can see that if we want to find an r -dimensional projection (for $r \leq k - 1$), we want the r top eigendirections, the subspace of the r eigenvectors with the r largest eigenvalues.

[4.6] [7 points] Implement the `transform` method in the LDA class to do supervised dimensionality reduction with LDA. `main.py lda-projection` then compares your supervised LDA projection to an unsupervised PCA projection on this distribution with 15 noise dimensions. You might want to change the `fit` function to support what you need for this (or not, you do you). Hand in your code and the produced figure.

Hint: It might be handy to use `scipy.linalg.solve_triangular` or `scipy.linalg.eigh`.

Answer: TODO