CPSC 540 Assignment 3 (due February 27)

Density Estimation and Project Proposal

1 Discrete and Gaussian Variables

1.1 MLE for General Discrete Distribution

Consider a density estimation task, where we have two variables ($d = 2$) that can each take one of k discrete values. For example, we could have

$$X = \begin{bmatrix} 1 & 3 \\ 4 & 2 \\ \text{k} & 3 \\ 1 & \text{k-1} \end{bmatrix}.$$

The likelihood for example x^i under a general discrete distribution would be

$$p(x_1^i, x_2^i|\Theta) = \theta_{x_1^i, x_2^i},$$

where θ_{c_1,c_2} gives the probability of x_1 being in state c_1 and x_2 being in state c_2, for all the k^2 combinations of the two variables. In order for this to define a valid probability, we need all elements θ_{c_1,c_2} to be non-negative and they must sum to one, $\sum_{c_1=1}^{k}\sum_{c_2=1}^{k}\theta_{c_1,c_2} = 1$.

1. Given n training examples, derive the MLE for the k^2 elements of Θ.
2. Because of the sum-to-1 constraint, there are only (k^2-1) degrees of freedom in the discrete distribution, and not k^2. Derive the MLE for this distribution assuming that

$$\theta_{k,k} = 1 - \sum_{c_1=1}^{k}\sum_{c_2=1}^{k}\mathcal{I}[c_1 \neq k, c_2 \neq k]\theta_{c_1,c_2},$$

   so that the distribution only has $(k^2 - 1)$ parameters.
3. If we had separate parameter θ_{c_1} and θ_{c_2} for each variables, a reasonable choice of a prior would be a product of Dirichlet distributions,

$$p(\theta_{c_1}, \theta_{c_2}) \propto \theta_{c_1}^{\alpha_{c_1}-1}\theta_{c_2}^{\alpha_{c_2}-1}.$$

   For the general discrete distribution, a prior encoding the same assumptions would be

$$p(\theta_{c_1,c_2}) \propto \theta_{c_1,c_2}^{\alpha_{c_1}+\alpha_{c_2}-2}.$$

   Derive the MAP estimate under this prior.

Hint: it is convenient to write the likelihood for an example i in the form

$$p(x^i|\Theta) = \prod_{c\in[k]^2}\theta_c^{\mathcal{I}[x^i=c]},$$

where c is a vector containing (c_1, c_2), $[x^i = c]$ evaluates to 1 if all elements are equal, and $[k]^2$ is all ordered pairs (c_1, c_2). You can use the Lagrangian to enforce the sum-to-1 constraint on the log-likelihood, and you may find it convenient to define $N_c = \sum_{i=1}^{n} \mathcal{I}[x^i = c]$.

Solution

1. Having n training examples $\mathbf{x} := (x^1, \ldots, x^n)$ where $x^i \in \{1, 2, \ldots, k\}^2$ with

$$\mathbb{P}(x_1^i = c_1, x_2^i = c_2) = \theta_{c1,c2}.$$

By denoting Θ the matrix containing all the parameters θ_{c_1,c_2}, we can write the likelihood for the i-th variable as

$$\mathbb{P}(x^i|\Theta) = \prod_{c\in[k]^2} \theta_c^{\mathcal{I}[x^i=c]}.$$

By assuming independence between the variables we conclude

$$\begin{aligned}\mathbb{P}(\mathbf{x}|\Theta) &= \prod_{i=1}^{n} \mathbb{P}(x^i|\Theta) \\ &= \prod_{i=1}^{n} \prod_{c\in[k]^2} \theta_c^{\mathcal{I}[x^i=c]}.\end{aligned}$$

Exchanging the order in the products and defining $N_c = \sum_{i=1}^{n} \mathcal{I}[x^i = c]$, we get

$$\mathbb{P}(\mathbf{x}|\Theta) = \prod_{c\in[k]^2} \theta_c^{N_c}.$$

Since the logarithm is an increasing function, the maximizer of this equation is the same as the maximizer of

$$\log(\mathbb{P}(\mathbf{x}|\Theta)) = \sum_{c\in[k]^2} N_c \log(\theta_c). \tag{1}$$

Since we want the parameters θ_c to represent probabilities, we have the constraint $\sum_{c\in[k]^2} \theta_c = 1$. Hence if we use Lagrange multipliers we conclude that for all $l \in [k]$ we have

$$\frac{\partial}{\partial\theta_l}(\log(\mathbb{P}(\mathbf{x}|\Theta))) = \lambda \frac{\partial}{\partial\theta_l}(\sum_{c\in[k]^2} \theta_c).$$

Derivating and solving for θ_l we conclude that

$$\theta_l = \frac{N_l}{\lambda}.$$

Inserting this result into the constraint we get

$$\lambda = \sum_{c\in[k]^2} N_c.$$

Hence the maximum likelihood estimation gives for each $l \in [k]^2$

$$\theta_l = \frac{N_l}{\sum_{c\in[k]^2} N_c}.$$

2.**Solve in a nicer way to exercise**

3. We want to find the MAP for $\mathbb{P}(\Theta|\mathbf{x})$ or $\log(\mathbb{P}(\Theta|\mathbf{x}))$. Using Bayes rule we get

$$\mathbb{P}(\Theta|\mathbf{x}) \propto \mathbb{P}(\mathbf{x}|\Theta)\mathbb{P}(\Theta).$$

Choosing the prior

$$\mathbb{P}(\Theta) = \prod_{(c_1,c_2)\in[k]^2} \theta_{c1,c2}^{\alpha_{c_1}+\alpha_{c_2}-2}.$$

We can easily see that

$$\log(\mathbb{P}(\Theta)) = \sum_{(c_1,c_2)\in[k]^2} (\alpha_{c_1}+\alpha_{c_2}-2)\log(\theta_{c_1,c_2}).$$

Combining this result with equation (1) we get that the log posterior is given by

$$\log(\mathbb{P}(\Theta|\mathbf{x})) = \underbrace{\sum_{(c_1,c_2)\in[k]^2} N_{c_1,c_2}\log(\theta_{c_1,c_2})}_{\log(\mathbb{P}(\mathbf{x}|\Theta))} + \underbrace{\sum_{(c_1,c_2)\in[k]^2} (\alpha_{c_1}+\alpha_{c_2}-2)\log(\theta_{c_1,c_2})}_{\log(\mathbb{P}(\Theta))}.$$

Using the sum to one constraint and Lagrange multipliers we conclude that for all $(l_1,l_2)\in[k]^2$ we must have

$$\frac{\partial}{\partial\theta_{l_1,l_2}}\Big(\sum_{(c_1,c_2)\in[k]^2} N_{c_1,c_2}\log(\theta_{c_1,c_2}) + \sum_{(c_1,c_2)\in[k]^2} (\alpha_{c_1}+\alpha_{c_2}-2)\log(\theta_{c_1,c_2})\Big) = \lambda\frac{\partial}{\partial\theta_{l_1,l_2}}\Big(\sum_{(c_1,c_2)\in[k]^2}\theta_{c_1,c_2}\Big)$$

Taking the derivatives and solving for θ_{l_1,l_2} we get

$$\theta_{l1,l2} = \frac{N_{l1,l2}+\alpha_{l_1}+\alpha_{l_2}-2}{\lambda}.$$

Plugging in this value into the sum to one constraint we conclude

$$\lambda = \sum_{(c_1,c_2)\in[k]^2} N_{c1,c2}+\alpha_{c_1}+\alpha_{c_2}-2.$$

Hence the MAP estimate is

$$\theta_{l1,l2} = \frac{N_{l1,l2}+\alpha_{l_1}+\alpha_{l_2}-2}{\sum_{(c_1,c_2)\in[k]^2} N_{c1,c2}+\alpha_{c_1}+\alpha_{c_2}-2}.$$

1.2 Generative Classifiers with Gaussian Assumption

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

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

where Θ represents the parameters of our model. To classify a new example $\hat{x}$, generative classifiers would use

$$\hat{y} = \underset{y \in \{1,2,\ldots,k\}}{\arg\max}\ p(\hat{x}|y = c, \Theta)p(y = c|\Theta),$$

where in our case the total number of classes k is 3 (The denominator $p(\hat{x}|\Theta)$ is irrelevant to the classification since it is the same for all y.) Modeling $p(y = c|\Theta)$ is eays: we can just use a k-state categorical distribution,

$$p(y = c|\Theta) = \theta_c,$$

where θ_c is a single parameter for class c. The maximum likelihood estimate of θ_c is given by n_c/n, the number of times we have $y^i = c$ (which we've called n_c) divided by the total number of data points n.

Modeling $p(x|y = c, \Theta)$ is the hard part: we need to know the *probability of seeing the feature vector x given that we are in class c.* This corresponds to solving a density estimation problem for each of the k possible classes. To make the density estimation problem tractable, we'll assume that the distribution of x given that $y = c$ is given by a $\mathcal{N}(\mu_c, \Sigma_c)$ Gaussian distribution for a class-specific μ_c and Σ_c,

$$p(x|y = c, \Theta) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_c|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu_c)^T\Sigma_c^{-1}(x - \mu_c)\right).$$

Since we are distinguishing between the probability under k different Gaussians to make our classification, this is called *Gaussian discriminant analysis* (GDA). In the special case where we have a constant $\Sigma_c = \Sigma$ across all classes it is known as *linear discriminant analysis* (LDA) since it leads to a linear classifier between any two classes (while the region of space assigned to each class forms a convex polyhedron as in k-means clustering). Another common restriction on the Σ_c is that they are diagonal matrices, since this only requires $O(d)$ parameters instead of $O(d^2)$ (corresponding to assuming that the features are independent univariate Gaussians given the class label). Given a dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^n$, where $x^i \in \mathbb{R}^d$ and $y^i \in \{1, \ldots, k\}$, the maximum likelihood estimate (MLE) for the μ_c and Σ_c in the GDA model is the solution to

$$\underset{\mu_1,\mu_2,\ldots,\mu_k,\Sigma_1,\Sigma_2,\ldots,\Sigma_k}{\arg\max} \prod_{i=1}^n p(x^i|y^i, \mu_{y^i}, \Sigma_{y^i}).$$

This means that the negative log-likelihood will be equal to

$$\begin{aligned} -\log p(X|y, \Theta) &= -\sum_{i=1}^n \log p(x^i|y^i|\mu_{y^i}, \Sigma_{y^i}) \\ &= \sum_{i=1}^n \frac{1}{2}(x^i - \mu_{y^i})^T\Sigma_{y^i}^{-1}(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^n \log|\Sigma_{y^i}| + \text{const.} \end{aligned}$$

1. Derive the MLE for the GDA model under the assumption of *common diagonal covariance* matrices, $\Sigma_c = D$ (d parameters). (Each class will have its own mean μ_c.)
2. Derive the MLE for the GDA model under the assumption of *individual scale-identity* matrices, $\Sigma_c = \sigma_c^2 I$ (k parameters).
3. It's painful to derive these from scratch, but you should be able to see a pattern that would allow other common restrictions. Without deriving the result from scratch (hopefully), give the MLE for the case of *individual full* covariance matrices, Σ_c ($O(kd^2)$ parameters).
4. When you run *example_generative* 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 ($\sim 36\%$). 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. Write a function *generativeGaussian* that fits a GDA model to this dataset (using individual full covariance matrices). Hand in the function and report the test set accuracy.

5. In this question we would like to replace the Gaussian distribution of the previous problem with the more robust multivariate-t distribution so that it isn't influenced as much by the noisy data. Unlike the previous case, we don't have a closed-form solution for the parameters. However, if you run *example_tdist* it generates random noisy data and fits a multivariate-t model (you will need to add the *minFunc* directory to the Matlab path for the demo to work). By using the *multivariateT* model, write a new function *generativeStudent* that implements a generative model that is based on the multivariate-t distribution instead of the Gaussian distribution. Report the test accuracy with this model.

Hints: you will be able to substantially simplify the notation in parts 1-3 if you use the notation $\sum_{i \in y_c}$ to mean the sum over all values i where $y^i = c$. Similarly, you can use n_c to denote the number of cases where $y_i = c$, so that we have $\sum_{i \in y_c} 1 = n_c$. Note that the determinant of a diagonal matrix is the product of the diagonal entries, and the inverse of a diagonal matrix is a diagonal matrix with the reciprocals of the original matrix along the diagonal. For part three you can use the result from class regarding the MLE of a general multivariate Gaussian. You may find it helpful to use the included *logdet.m* function to compute the log-determinant in more numerically-stable way.

1.3 Self-Conjugacy for the Mean Parameter

If x is distributed according to a Gaussian with mean μ,

$$x \sim \mathcal{N}(\mu, \sigma^2),$$

and we assume that μ itself is distributed according to a Gaussian

$$\mu \sim \mathcal{N}(\alpha, \gamma^2),$$

then the posterior $\mu|x$ also follows a Gaussian distribution.[1] Derive the form of the (Gaussian) distribution for $p(\mu|x, \alpha, \sigma^2, \gamma^2)$.

Hints: Use Bayes rule and use the $\propto$ sign to get rid of factors that don't depend on μ. You can "complete the square" to make the product look like a Gaussian distribution, e.g. when you have $\exp(ax^2 - bx + \text{const})$ you can factor out an a and add/subtract $(b/2a)^2$ to re-write it as

$$\begin{aligned}\exp\left(ax^2 - bx + const\right) &\propto \exp\left(ax^2 - bx\right) = \exp\left(a(x^2 - (b/a)x)\right) \\ &\propto \exp\left(a(x^2 - (b/a)x + (b/2a)^2)\right) = \exp\left(a(x - (b/2a))^2\right).\end{aligned}$$

Note that multiplying by factors that do not depend on μ within the exponent does not change the distribution. In this question you will want to complete the square to get the distribution on μ, rather than x. You may find it easier to solve thie problem if you parameterize the Gaussians in terms of their 'precision' parameters (e.g., $\lambda = 1/\sigma^2$, $\lambda_0 = 1/\gamma^2$) rather than their variances σ^2 and γ^2.

2 Mixture Models and Expectation Maximization

2.1 Semi-Supervised Gaussian Discriminant Analysis

Consider fitting a GDA model where some of the y^i values are missing at random. In particular, let's assume we have n labeled examples (x^i, y^i) and then another another t unlabeled examples (x^i). This is a special

[1] We say that the Gaussian distribution is the 'conjugate prior' for the Gaussian mean parameter (we'll formally discuss conjugate priors later in the course). Another reason the Gaussian distribution is important is that is the only (non-trivial) continuous distribution that has this 'self-conjugacy' property.

case of *semi-supervised learning*, and fitting generative models with EM is one of the oldest semi-supervised learning techqniues. When the classes exhibit clear structure in the feature space, it can be very effective even if the number of labeled examples is very small.

1. Derive the EM update for fitting the parameters of a GDA model (with individual full covariance matrices) in the semi-superivsed setting where we have n labeled examples and t unlabeled examples.

2. If you run the demo *example_SSL*, it will load a variant of the dataset from the previous question, but where the number of labeled examples is small and a large number of unlabeled examples are available. The demo first fits a KNN model and then a generative Gaussian model (once you are finished Question 1). Because the number of labeled examples it quite small, the performance is worse than in Question 2. Write a function *generativeGaussianSSL* that fits the generative Gaussian model of the previous question using EM to incorporate the unlabeled data. Hand in the function and report the test error when training on the full dataset.

3. Repeat the previous part, but using the "hard"-EM algorithm where we explicitly classify all the unlabeled examples. How does this change the performance and the number of iterations?

Hint: for the first question most of the work has been done for you in the EM notes on the course webpage. You can use the result (**) and the update of θ_c from those notes, but you will need to work out the update of the parameters of the Gaussian distribution $p(x^i|y^i, \Theta)$.

Hint: for the second question, although EM often leads to simple updates, implementing them correctly can often be a pain. One way to help debug your code is to compute the observed-data log-likelihood after every iteration. If this number goes down, then you know your implementation has a problem. You can also test your updates of sets of variables in this way too. For example, if you hold the μ_c and Σ_c fixed and only update the θ_c, then the log-liklihood should not go down. In this way, you can test each of combinationso of updates on their to make sure they are correct.

2.2 Mixture of Bernoullis

The function *example_Bernoulli* loads a binarized version of the MNIST dataset and fits a density model that uses an independent Bernoulli to model each feature. If the reports the average NLL on the test data and shows 4 samples generated from the model. Unfortunately, the test NLL is infinity and the samples look terrible.

1. To address the problem that the average NLL is infinity, modify the *densityBernoulli* function to implement Laplace smoothing based on an extra argument α. Hand in the code and report the average NLL with $\alpha = 1$.

2. Write a new function implementing the mixture of Bernoullis model with Laplace smoothing of the θ values (note that Laplace smoothing only change the M-step). Hand in the code and report the average NLL with $\alpha = 1$ and $k = 10$ for a particular run of the algorithm, as well as 4 samples from the model and 4 of the cluster images.