Assignment #2

In this assignment, we'll fit both generative and discriminative models to the MNIST dataset of handwritten numbers. Each datapoint in the MNIST [`http://yann.lecun.com/exdb/mnist/`] dataset is a 28x28 black-and-white image of a number in $\{0 \dots 9\}$, and a label indicating which number.

MNIST is the 'fruit fly' of machine learning - a simple standard problem useful for comparing the properties of different algorithms. Python code for loading and plotting MNIST is attached.

You can use whichever programming language you like, and libraries for loading and plotting data. You'll need to write your own initialization, fitting, and prediction code. You can use automatic differentiation in your code, but must still answer the gradient questions.

For this assignment, we'll *binarize* the dataset, converting the grey pixel values to either black or white (0 or 1) with > 0.5 being the cutoff. When comparing models, we'll need a training and test set. Use the first 10000 samples for training, and another 10000 for testing. This is all done for you in the starter code. Hint: Also build a dataset of only 100 training samples to use when debugging, to make loading and training faster.

Problem 1 (Basic Naïve Bayes, 10 points)

In this question, we'll fit a naïve Bayes model to the MNIST digits using maximum likelihood. Naïve Bayes defines the joint probability of the each datapoint $\mathbf{x}$ and its class label c as follows:

$$p(\mathbf{x}, c|\boldsymbol{\theta}, \pi) = p(c|\pi)p(\mathbf{x}|c, \theta_c) = p(c|\pi)\prod_{d=1}^{784} p(x_d|c, \theta_{cd}) \tag{1}$$

For binary data, we can use the Bernoulli likelihood:

$$p(x_d|c, \theta_{cd}) = Ber(x_d|\theta_{cd}) = \theta_{cd}^{x_d}(1-\theta_{cd})^{(1-x_d)} \tag{2}$$

Which is just a way of expressing that $p(x_d = 1|c, \theta_{cd}) = \theta_{cd}$.

For $p(c|\pi)$, we can just use a categorical distribution:

$$p(c|\pi) = Cat(c|\pi) = \pi_c \tag{3}$$

Note that we need $\sum_{i=0}^{9} \pi_i = 1$.

(a) Derive the *maximum likelihood estimate* (MLE) for the class-conditional pixel means $\boldsymbol{\theta}$. Hint: We saw in lecture that MLE can be thought of as 'counts' for the data, so what should $\hat{\theta}_{cd}$ be counting?

(b) Derive the *maximum a posteriori* (MAP) estimate for the class-conditional pixel means $\boldsymbol{\theta}$, using a Beta(2, 2) prior on each θ. Hint: it has a simple final form, and you can ignore the Beta normalizing constant.

(c) Fit $\boldsymbol{\theta}$ to the training set using the MAP estimator. Plot $\boldsymbol{\theta}$ as 10 separate greyscale images, one for each class.

(d) Derive the predictive log-likelihood $\log p(c|\mathbf{x}, \boldsymbol{\theta}, \pi)$ for a single training image.

(e) Given parameters fit to the training set, and $\pi_c = \frac{1}{10}$, report both the average predictive log-likelihood per datapoint, $\frac{1}{N}\Sigma_{i=1}^{N} \log p(x_i, c_i|\theta, \pi)$ and the predictive accuracy on both the training and test set. The predictive accuracy is defined as the fraction of examples where the true class $t = \text{argmax}_c\, p(c|\mathbf{x}, \theta, \pi)$.

The takeaway of this question is that we can automatically derive a learning algorithm just by first defining a joint probability!

Problem 2 (Advanced Naïve Bayes, 10 points)

One of the advantages of generative models is that they can handle missing data, or be used to answer different sorts of questions about the model.

(a) True or false: Given our model's assumptions, any two pixels x_i and x_j where $i \neq j$ are independent given c.

(b) True or false: Given our model's assumptions, any two pixels x_i and x_j where $i \neq j$ are independent when marginalizing over c.

(c) Using the parameters fit in question 1, produce random image samples from the model. That is, randomly sample and plot 10 binary images from the marginal distribution $p(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\pi})$. Hint: Use ancestral sampling.

(d) Derive $p(\mathbf{x}_{bottom}|\mathbf{x}_{top}, \boldsymbol{\theta}, \pi)$, the joint distribution over the bottom half of an image given the top half, conditioned on your fit parameters.

(e) Derive $p(\mathbf{x}_{i \in bottom}|\mathbf{x}_{top}, \boldsymbol{\theta}, \boldsymbol{\pi})$, the marginal distribution of a single pixel in the bottom half of an image given the top half, conditioned on your fit parameters.

(f) For 20 images from the training set, plot the top half the image concatenated with the marginal distribution over each pixel in the bottom half.

Problem 3 (Logistic Regression, 10 points)

Now, we'll fit a simple predictive model using gradient descent. Our model will be multiclass logistic regression:

$$p(c|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=0}^{9} \exp(\mathbf{w}_{c'}^T \mathbf{x})} \tag{4}$$

You can ignore biases for this question.

(a) How many parameters does this model have?

(b) Since class probabilities must sum to one, this imposes constraints on the predictions of our model. What is the smallest number of parameters we could use to write an equally expressive model with a different parameterization?

(c) Derive the gradient of the predictive log-likelihood w.r.t. $\mathbf{w}$: $\nabla_{\mathbf{w}} \log p(c|\mathbf{x}, \mathbf{w})$

(d) Code up a gradient-based optimizer of your choosing, it can be just vanilla gradient descent, and use it to optimize $\mathbf{w}$ to maximize the log-likelihood of the training set, and plot the resulting parameters using one image per class. Since this objective is concave, you can initialize at all zeros. Using automatic differentiation is permitted, so you can use autograd to get gradients for use by your optimizer, and using minibatches is optional. However, you are not permitted to use optimizers which come built in to packages! Hint: Use `scipy.logsumexp` or its equivalent to make your code more numerically stable.

(e) Given parameters fit to the training set, report both the average predictive log-likelihood per datapoint, and the predictive accuracy on both the training and test set. How does it compare to Naïve Bayes?

Problem 4 (Unsupervised Learning, 10 points)

Another advantage of generative models is that they can be trained in an unsupervised or semi-supervised manner. In this question, we'll fit the Naïve Bayes model without using labels. Since we don't observe labels, we now have a *latent variable model*. The probability of an image under this model is given by the marginal likelihood, integrating over c:

$$p(\mathbf{x}|\theta,\pi) = \sum_{c=1}^{k} p(\mathbf{x},c|\theta,\pi) = \sum_{c=1}^{k} p(c|\pi) \prod_{d=1}^{784} p(x_d|c,\theta_{cd}) = \sum_{c=1}^{k} Cat(c|\pi) \prod_{d=1}^{784} Ber(x_d|\theta_{cd}) \tag{5}$$

It turns out that this gives us a mixture model! This model is sometimes called a "mixture of Bernoullis", although it would be clearer to say "mixture of products of Bernoullis". Again, this is just the same Naïve Bayes model as before, but where we haven't observed the class labels c. In fact, we are free to choose K, the number of mixture components.

(a) Given K, how many parameters does this model have?

(b) How many ways can we permute the parameters of the model $\boldsymbol{\theta}, \boldsymbol{\pi}$ and get the same marginal likelihood $p(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\pi})$? Hint: switching any two classes won't change the predictions made by the model about $\mathbf{x}$.

(c) Derive the gradient of the log marginal likelihood with respect to θ: $\nabla_\theta \log p(\mathbf{x}|\theta,\pi)$

(d) For a fixed $\pi_c = \frac{1}{K}$ and K = 30, fit θ on the training set using gradient based optimization. Note: you can't initialize at all zeros – you need to break symmetry somehow, which is done for you in starter code. Starter code reduces this problem to correctly coding the optimization objective. Plot the learned θ. How do these cluster means compare to the supervised model?

(e) For 20 images from the training set, plot the top half the image concatenated with the marginal distribution over each pixel in the bottom half. Hint: You can re-use the formula for $p(\mathbf{x}_{i\in bottom}|\mathbf{x}_{top},\theta,\pi)$ from before. How do these compare with the image completions from the supervised model?