HOMEWORK 2 - V1

STA414/STA2104 WINTER 2019

University of Toronto

VERSION HISTORY: V0 → V1: ADD HINT TO Q4

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 and R codes 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.

1. Fitting a Naïve Bayes Model, 25 pts. 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:

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

Here, $\boldsymbol{\theta}$ is a matrix of probabilities for each pixel and each class, so its total size is 784×10.

For binary data, we can use the Bernoulli likelihood:

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

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:

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

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

2. Generating from a Naïve Bayes Model, 25 pts. Defining a joint probabilistic model over the data lets us generate new data, and also lets us answer all sorts of queries about the data.

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

(b) True or false: Given this 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: first sample c.

(d) 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. 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. Hint: you'll have to marginalize over c.

(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. i.e. the bottom half of the image should use grayscale to represent the marginal probability of each pixel being on.

3. Logistic Regression, 25 pts. Now, we'll fit a simple predictive model using gradient descent. Our model will be multiclass logistic regression:

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

Omit bias parameters for this question.

(a) How many parameters does this model have? 10*784

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

(c) 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. You can use the gradient ascent (since it is a maximization) updates as covered in class or use automatic differentiation (autograd). 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. Also, try to avoid nested `for` loops, and instead use matrix operations to keep your code fast.

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

4. Unsupervised Learning, 25 pts. In this problem, we will experiment with two clustering algorithms: (i) k-means, and (ii) EM algorithm for Gaussian mixtures. In what follows, N denotes the number of data points, D denotes the dimension of each data point, $x_i \in \mathbb{R}^D$ denotes a single data point, K denotes the number of clusters, and $x_i \sim p(x|\mathcal{C}_k)$ for $k = 1, ..., K$ denotes the data generating distribution.

(a) First, we will generate some data for this problem. Set the number of points $N = 400$, their dimension $D = 2$, and the number of clusters $K = 2$, and generate data from the distribution $p(x|\mathcal{C}_k) = \mathcal{N}(\mu_k, \Sigma_k)$. Sample 200 data points for $\mathcal{C}_1$ and 200 for $\mathcal{C}_2$, with

$$\mu_1 = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}, \mu_2 = \begin{bmatrix} 6.0 \\ 0.1 \end{bmatrix} \text{ and } \Sigma_1 = \Sigma_2 = \begin{bmatrix} 10 & 7 \\ 7 & 10 \end{bmatrix}$$

Here, $N = 400$. Since you generated the data, you already know which sample comes from which class. Make a scatter plot of the data points showing the true cluster assignment of each point either using different color codes or shape or both.

(b) Now, we assume that the true class labels are not known. Implement the k-means algorithm for this problem. Write three functions: `cost`, `km_e_step`, and `km_m_step` as given in the lecture (Here, `km_` means k-means). Identify the correct arguments, and the order to run them. Initialize the algorithm with

$$\hat{\mu}_1 = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}, \hat{\mu}_2 = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix} \tag{4.1}$$

and run it until convergence. Show the resulting cluster assignments on a scatter plot either using different color codes or shape or both. Also plot the cost vs. the number of iterations. Report your misclassification error.

(c) Next, implement the EM algorithm for Gaussian mixtures. Write four functions: `normal_density`, `log-likelihood`, `em_e_step`, and `em_m_step` as given in the lecture. Identify the correct arguments, and the order to run them. Initialize the algorithm with means as in (4.1), covariances with $\hat{\Sigma}_1 = \hat{\Sigma}_2 = I$, and $\hat{\pi}_1 = \hat{\pi}_2$. Run the algorithm until convergence and show the resulting cluster assignments on a scatter plot either using different color codes or shape or both. Also plot the log-likelihood vs. the number of iterations. Report your misclassification error.

(d) Comment on your findings, i.e., resulting cluster assignments, number of iterations until convergence, etc. Experiment with different data realizations (generate new data), run your algorithms, and summarize your findings. Does the algorithm performance depend on different realizations of data? *Hint: In most cases, k-means should fail to identify the correct cluster labels due to the covariance structure. There may be realizations that EM also fails to find the clusters correctly but in general it should work better than k-means.*