CPSC 440/540 Machine Learning (Jan-Apr 2023)
Assignment 2 – due Friday February 17th 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.

1 Bernoulli Inference [18 points]

Consider a Bernoulli distribution with $\theta = 0.17$.

(a) [2 points] Suppose we generate 10 IID samples $\{x^1, x^2, ..., x^{10}\}$ according to this model. What is the probability that all 10 samples are equal to 0?

Answer: TODO

(b) [2 points] What is the probability that exactly one sample is equal to 1, and the other 9 are equal to 0?

Answer: TODO

(c) [2 points] Suppose we need to make some decisions based on a bunch of points that represent this distribution (Bernoulli(θ)), rather than directly reasoning about the distribution itself. We'd like to gather 1,000 points to represent the distribution. Would it make sense to chose the mode, the "most likely" set of points to be seen, to make decisions? Explain why or why not.

Answer: TODO.

(d) [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. Hand in your code.

Answer: TODO

(e) [2 points] Consider a game where you get \$1 if a sample from this distribution is 0, and lose \$5 if the sample is 1. What is the expected \$ value you get from playing this game?

Answer: TODO

(f) [4 points] Supposing 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 t IID sample and f gives the \$ value for that sample (either 1 or -5).

Implement this approximation, and use the code in `q_1f` 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

(g) [2 points] Suppose that you wanted to compute the expected number of samples from this Bernoulli that you would need to generate before seeing 10 values of 1. It's possible to compute this expectation exactly, but suppose we really like sampling and are too lazy to do the exact calculation. Explicitly describe a way to estimate this quantity from samples, similar to the one from the previous part.

Answer: TODO

(h) [2 points] Suppose that the *rng.random()* function were found to be broken in a new version of numpy, in the sense that the numbers it returned were found to not look like uniform samples.[1] Suppose, though, that the *rng.normal()* function, which generates samples from a standard normal distribution, still worked well. Describe a way to generate samples from a Bernoulli distribution that relies on the *normal()* function instead of the *random()* function. *Hint: You may want to look up the CDF (and its inverse, the "quantile" function) of the standard normal distribution.*

Answer: TODO

[1] Some published research works have had to be revised due to bad random number generators.

2 Learning with Bernoullis [5 points]

We often use Bernoulli distributions as parts of more complicated distributions. In these cases we often to need to maximize a weighted log-likelihood of the form

$$f(\theta) = \sum_{i=1}^{n} v^i \log p(x^i \mid \theta),$$

where each v^i is a non-negative weight for example i. Derive the MLE for this model when we use a Bernoulli likelihood for $p(x^i \mid \theta)$. You can take as given that f is concave (so that any stationary point is a global optimum).

Answer: TODO

3 Naive Bayes [33 points]

If you run `main.py 3`, it will load training and test data for MNIST digits 3 and 5, discretized into binary values. It will then discretize the features into binary values, and fit a "naive" naive Bayes model. In particular, the naive naive Bayes model is a generative classifier that assumes $p(x_1, x_2, \ldots, x_d, y)$ is product of Bernoullis. This model has a very high test error (46.9%), since the features do not affect the predictions.

(a) [3 points] The regular naive Bayes model discussed in class writes the joint probability of a dataset $(\mathbf{X}, y)$ as

$$p(\mathbf{X}, y) = \prod_{i=1}^{n} \left[p(y^i) \prod_{j=1}^{d} p(x_j^i \mid y^i) \right]$$
$$= \prod_{i=1}^{n} \left[\theta^{y^i} (1-\theta)^{1-y^i} \prod_{j=1}^{d} \left[\theta_{jy^i}^{x_j^i} (1-\theta_{jy^i})^{1-x_j^i} \right] \right],$$

using Bernoullis to parameterize $p(y^i)$ and each conditionals $p(x_j \mid y)$. Show how to derive the MLE for any particular parameter θ_{jc}. You can assume here that the log-likelihood is concave (so that any stationary point is a global optimum).

Answer: TODO

(b) [2 points] Even though the naive Bayes likelihood has many parameters, the MLE for a particular parameter θ_{jc} does not depend on θ or any any $\theta_{j'c'}$ with $j \neq j'$ and $c' \neq c$. Explain why these terms do not appear in the MLE. (Don't use the probabilistic concept of "independence" in your answer. Instead explain how the form of the log-likelihood makes them not depend on each other.)

Answer: TODO

(c) [8 points] Implement a standard naive Bayes classifier with this parameterization in `naive_bayes.py`. Include Laplace smoothing, since otherwise you'll get some NaNs. Hand in your code, and the output of `python main.py 3c`, which tests with several different choices for the amount of Laplace smoothing.

Answer: TODO

(d) [4 points] Naive Naive Bayes was *way* too naive; Naive Bayes still doesn't do *that* well. What if we removed the independence assumption and just went for a full tabular parameterization in a "Galaxy Brain Bayes" model

$$p(\mathbf{X}, y) = \prod_{i=1}^{n} \left[p(y^i) p(x^i \mid y^i) \right],$$

where $p(y)$ is Bernoulli and $p(x \mid y)$ is a full tabular distribution, with Laplace smoothing (adding one pseudocount per possible x value). Without actually implementing it (unless you want to), what would the test and training error of this model be on the current problem? Give your reasoning; be explicit about what the predictions will be.

Answer: TODO

How much better can we expect to get? `python main.py 3-logistic` will fit a logistic regression model, which gets 4.6% error. (And CNN-based models, and some others, can get almost zero error on this problem.) So, let's try one more way to make naive Bayes less naive, which will actually improve its performance.

It seems reasonable that there would be clusters in the classes. For example, there might several different general ways to draw the digit `3`. Instead of assuming that the features are independent given the class label, we might instead assume they are independent given the cluster that they are in. Consider a *vector-quantized naive Bayes* (VQNB) that implements this idea:

- It clusters the *examples associated with each digit* into k clusters, using k-means clustering (k clusters for each class, $2k$ 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 "3" clusters or the k "5" clusters.
- 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

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

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

(e) [10 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.

Hint: $p(y)$ you can handle just as before. $p(z \mid y)$ is a categorical variable; we derived the MLE of these kinds of variables in class. $p(x_j \mid y, z)$ is Bernoulli; you'll have one Bernoulli parameter for each (y, z) pair, which it might be convenient to organize in a $2 \times k$ array.

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 may not be able to exactly match the performance of logistic regression.

Answer: TODO

(f) [2 points] What is the computational cost of making a prediction with this model?

Answer: TODO

(g) [2 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 10 images). There's a code stub in there for plotting it, just fill that out.

Answer: TODO

(h) [2 points] The logistic regression model and VQNB model obtain similar accuracies, and both give estimates of the probability for $p(y^i \mid x_1^i, x_2^i, \ldots, x_d^i)$. Give an example of something we could do with the VQNB model that we couldn't do with the logistic regression model.

Answer: TODO

4 Neural Networks [20 points]

4.1 Hand-Made, Artisan Neural Networks [4 points]

Suppose that we train a neural network with sigmoid activations and one hidden layer and obtain the following parameters (assume that we don't use any bias variables):

$$W = \begin{bmatrix} -2 & 2 & -1 \\ 1 & -2 & 0 \end{bmatrix}, \quad v = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

Assuming that we are doing regression, for a training example with features $(x^i)^T = \begin{bmatrix} -3 & -2 & 2 \end{bmatrix}$, what are the values in this network of the hidden units z^i, activations $a^i = h(z^i)$, and prediction $\hat{y}^i$?

Answer: TODO

4.2 Neural Network Tuning - Regression [8 points]

The file `example_nnet.jl` runs a stochastic gradient method to train a neural network on the *basisData* dataset from a previous assignment. However, in its current form it doesn't fit the data very well. Modify the training procedure and model to improve the performance of the neural network. Hand in your plot after changing the code to have better performance, and list the changes you made.

Hint: There are many possible strategies you could take to improve performance. Below are some suggestions, but note that the some will be more effective than others:

- *Changing the network structure (`hidden_layer_sizes` is a list giving the number of hidden units in each layer).*
- *Changing the training procedure (you can change the stochastic gradient step-size, use decreasing step sizes, use mini-batches, run it for more iterations, add momentum, switch to* findMin*, use Adam, and so on).*
- *Transform the data by standardizing the features, standardizing the targets, and so on.*
- *Add regularization (L2-regularization, L1-regularization, dropout, and so on).*
- *Change the initialization.*
- *Add bias variables.*
- *Change the loss function or the non-linearities (right now it uses squared error and* tanh *to introduce non-linearity).*
- *Use mini-batches of data, possibly with batch normalization.*

Answer: TODO

4.3 Neural Network Tuning - Classification [8 points]

The file `example_mnist35_nnet.jl` runs a stochastic gradient method to train a neural network on the '3' and ''5' images from the naive Bayes question (it uses the squared error for training and classifies by taking the sign of the prediction). Modify the training procedure and model so that the neural network has a better error on the test set than the 0.03 achieved by logistic regression. List the changes you made and the best test performance that you were able to obtain.

Answer: TODO

5 Very-Short Answer Questions [24 points]

Answer these questions **briefly**: at most a couple of lines. If you're super-verbose, you'll lose points!

Each is worth [2 points].

(a) List 5 different inference tasks you might do with a (single variable) Bernoulli model.

Answer: TODO

(b) What is a reason we might want to sample from a distribution?

Answer: TODO

(c) Suppose we have $p(x) \propto \alpha g(x)$ for some continuous variable x, some positive constant α, and some non-negative function g. Give the form of the distribution if we do not use the $\propto$ sign to hide the proportionality constant.

Answer: TODO

(d) What is the advantage of using a discriminative model for supervised learning, rather than the generative approach of treating supervised learning as special case of density estimation.

Answer: TODO

(e) Give a choice of the number of hidden units, the activation function, and likelihood in a neural network with a single hidden layer such that MLE in the model makes the same predictions as the MLE logistic regression model.

Answer: TODO

(f) Describe an experimental setup where you might see a double descent curve when fitting a logistic regression model with L2-regularization. You can assume you have a way to generate new relevant features. Caution: the global minimum is unique in this model so the double descent would not come from choosing among multiple global minima.

Answer: TODO

(g) Consider a neural network with L hidden layers, where each layer has at most k hidden units and we have c labels in the final layer (and the number of features is d). What is the cost of prediction with this network?

Answer: TODO

(h) What is one advantage of using automatic differentiation and one disadvantage?

Answer: TODO

(i) Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification?

Answer: TODO

(j) Why do "vanilla" autoencoders have a bottleneck layer?

Answer: TODO

(k) We discussed multi-label classification using neural networks and a product of Bernoullis model. The product of Bernoullis model assumes the class labels are independent, so why might this model make predictions that seem to capture dependencies between the random variables?

Answer: TODO

(l) Why can fully-convolutional networks segment images of different sizes?

Answer: TODO