CSE 151: Machine learning **Winter 2019**

Homework 4

Submission instructions:

- Please type up your solutions.
- If a problem asks for a numerical answer, you need only provide this answer. There is no need to show your work, unless you would like to.
- Upload the PDF file for your homework to `gradescope` by 6pm on Tuesday February 5.

Part A: Short-answer questions on regression

1. *Example of regression with one predictor variable.* Consider the following simple data set of four points (x, y):
$$(1,1),(1,3),(4,4),(4,6).$$
   (a) Suppose you had to predict y without knowledge of x. What value would you predict? What would be its mean squared error (MSE) on these four points?

   (b) Now let's say you want to predict y based on x. What is the MSE of the linear function $y = x$ on these four points?

   (c) Find the line $y = ax + b$ that minimizes the MSE on these points. What is its MSE?

2. *Lines through the origin.* Suppose that we have data points $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$, where $x^{(i)}, y^{(i)} \in \mathbb{R}$, and that we want to fit them with a line that passes through the origin. The general form of such a line is $y = ax$: that is, the sole parameter is $a \in \mathbb{R}$.

   (a) The goal is to find the value of a that minimizes the squared error on the data. Write down the corresponding loss function.

   (b) Using calculus, find the optimal setting of a.

3. We have a data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$, where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$. We want to express y as a linear function of x, but the error penalty we have in mind is not the usual squared loss: if we predict $\widehat{y}$ and the true value is y, then the penalty should be the absolute difference, $|y - \widehat{y}|$. Write down the loss function that corresponds to the total penalty on the training set.

4. We have n data points in $\mathbb{R}^d$ and we want to compute all pairwise dot products between them. Show that this can be achieved by a *single* matrix multiplication.

Part B: Programming project on generative modeling

In class, we mentioned the MNIST data set of handwritten digits. In this problem, you will build a classifier for this data, by modeling each class as a multivariate (784-dimensional) Gaussian.

- Download the Jupyter notebook `generative-mnist.ipynb` from the course website and unzip it. This will help you by loading in the MNIST data set.
  - Look over the notebook to see what it is doing, and then run it, one cell at a time.
  - Make sure you understand the form in which the training and test data are stored.
  - Towards the end of the notebook, there is also a helper function that displays digits.
- Split the training set into two pieces – a training set of size 50000 (say), and a separate *validation set* of size 10000.
- Now fit a Gaussian generative model to the training data of 50000 points.
  - Determine the class probabilities: what fraction π_0 of the training points are digit 0, for instance? Call these values $\pi_0, \ldots, \pi_9$.
  - Fit a Gaussian to each digit, by finding the mean and the covariance of the corresponding data points. Let the Gaussian for the jth digit be $P_j = N(\mu_j, \Sigma_j)$. Note that μ_j will be a 784-dimensional vector, and Σ_j will be a 784×784 matrix.

  Using these two pieces of information, you can classify new images x using Bayes' rule: simply pick the digit j for which $\pi_j P_j(x)$ is largest.
- One last step is needed: it is important to smooth the covariance matrices, and the usual way to do this is to add in cI, where c is some constant and I is the identity matrix. Use the validation set to help you choose the right value of c: that is, choose the value of c for which the resulting classifier makes the fewest mistakes on the validation set.
- There are some important details of *numerical precision* that you will need to attend to. In 784-dimensional space, all probabilities $P_j(x)$ will likely be miniscule, and this can produce all sorts of trouble due to underflow errors. It is better to work with log-probabilities: -1000 is easier to deal with than e^{-1000}. This means that you should classify a point x by picking the j that maximizes $\log \pi_j + \log P_j(x)$. Fortunately, the Python `multivariate_normal` package will directly compute $\log P_j(x)$ for you.

To turn in:

1. Pseudocode for your training procedure, making it clear how the validation set was created and used.
2. Did you use a single value of c for all ten classes, or separate values for each class? What value(s) of c did you get?
3. What was the error rate on the MNIST test set?
4. Out of the misclassified test digits, pick five at random and display them. For each instance, list the posterior probabilities $\Pr(y|x)$ of each of the ten classes.