CS5793: Artificial Intelligence II
Assignment 2

Prof. Christopher Crick

1 Naive Bayes

Download the MNIST database files from `http://yann.lecun.com/exdb/mnist/`. Take a look at the file formats, explained on that page. Python makes it very easy to read in and manipulate data.

```
training_images_file = open('train-images-idx3-ubyte','rb')
training_images = training_images_file.read()
training_images_file.close()
```

The 'rb' parameter tells Python to open the file for reading in binary mode. Binary mode has no effect on Unix systems, but is still a good idea to indicate the expected file contents (on Windows and Mac systems, opening a file NOT in binary mode fixes all of the newline characters, which is not something you want unless you are actually dealing with text – which, to be fair, you often are). This code reads in the contents of the file as a string, which we can convert to a list of bytes by using the bytearray function. We need to do this because the Numpy array constructors expect lists, not strings.

```
training_images = bytearray(training_images)
```

The MNIST files have some bytes up front devoted to magic numbers and bookkeeping information, so we can just throw those away.

```
training_images = training_images[16:]
```

Now, turn this into a Numpy array and shape it properly, N rows and D features. You can even test to see that you have extracted it properly by pulling out a row, shaping it to a 28x28 matrix, and plotting it.

Notice that these images are not binary. This dataset has been anti-aliased, and the byte values range from 0 to 255, not $x_{nd} \in \{0, 1\}$. Fix that by using a threshold. (Note: operations applied to Numpy arrays are almost always applied elementwise. So it is almost never necessary to explicitly set up for loops to operate on them, which makes operations faster to code, faster to execute, and easier to understand. Avoid for loops.)

With your training images and labels, create a naive Bayes classifier. This will yield an array of θ_{kd} values, with each element $\theta_{kd} = p(x_d = 1|k)$. Use this classifier (including a uniform Dirichlet prior) on the MNIST test set and report the classification accuracy. When computing likelihoods, remember that you don't care about the actual value, you only care about which value is the maximum. So it is perfectly okay not to normalize, and even better to compute and compare the log likelihoods instead (addition does not risk numerical instability problems, unlike multiplication).

2 Naive Bayes Gaussian

Using the original (non-thresholded, 0-255 scale) MNIST training set, construct training and test sets to use for binary classification. Extract 1000 representatives of class 5 and 1000 representatives of numbers that are not 5, chosen uniformly at random from among the other nine options. Assign 90% of these selections to your training set and 10% to your test set. Create a new naive Bayes classifier that uses Gaussian kernels rather than a categorical distribution. The model will now have one more parameter per class – a μ_{kd} for each pixel, given the category, and a single variance σ_k^2 for the entire category. However, there are only two classes – "yes, it's a 5" and "no, it is not a 5".

The probability density function for a Gaussian distribution is

$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Using this model, compute the true positive vs false positive rates for the following situations:

- Type I errors are 5 times as costly as Type II errors
- Type I errors are twice as costly as Type II errors
- Both types of errors are equally costly
- Type II errors are twice as costly as Type I errors
- Type II errors are 5 times as costly as Type I errors

Plot the ROC curve for this model.

3 More nearest neighbors

Using the original (non-thresholded, 0-255 scale) MNIST training set, cut it down significantly to make the computational complexity of a high-dimensional nearest-neighbors search a little more tractable. Create a training set consisting of 200 examples of each of three classes, the numerals 1, 2 and 7. Don't bother with a KD tree – remember that in high-dimensional spaces, a KD tree is no better than brute force search. This is why we are keeping the size down to 600. Go ahead and implement a brute force nearest neighbors search, computing the 784-dimensional Euclidean distance:

$d(x_i, x_j) = \sqrt{\sum_{d=1}^{784} (x_{id} - x_{jd})^2}$

Conduct a 5-fold cross validation on your training set, testing the NN classifier on five candidate classifier models, those with $K = \{1, 3, 5, 7, 9\}$. Determine the best candidate model.

Create a test data set from the MNIST test data, consisting of 50 examples of each of the three classes we are considering. Using the NN model selected by your model validation, classify your test set. Compare the accuracy of your model on your test set with its cross-validation accuracy.

For each of the three classes, plot some test set examples which your classifier identified correctly and some for which it failed. Can you identify any patterns which seem likely to lead to failure?