CS360 Project 2.1, Spring 2018

Due Date: March 30th, 2018, 11:59PM PST

7 Points

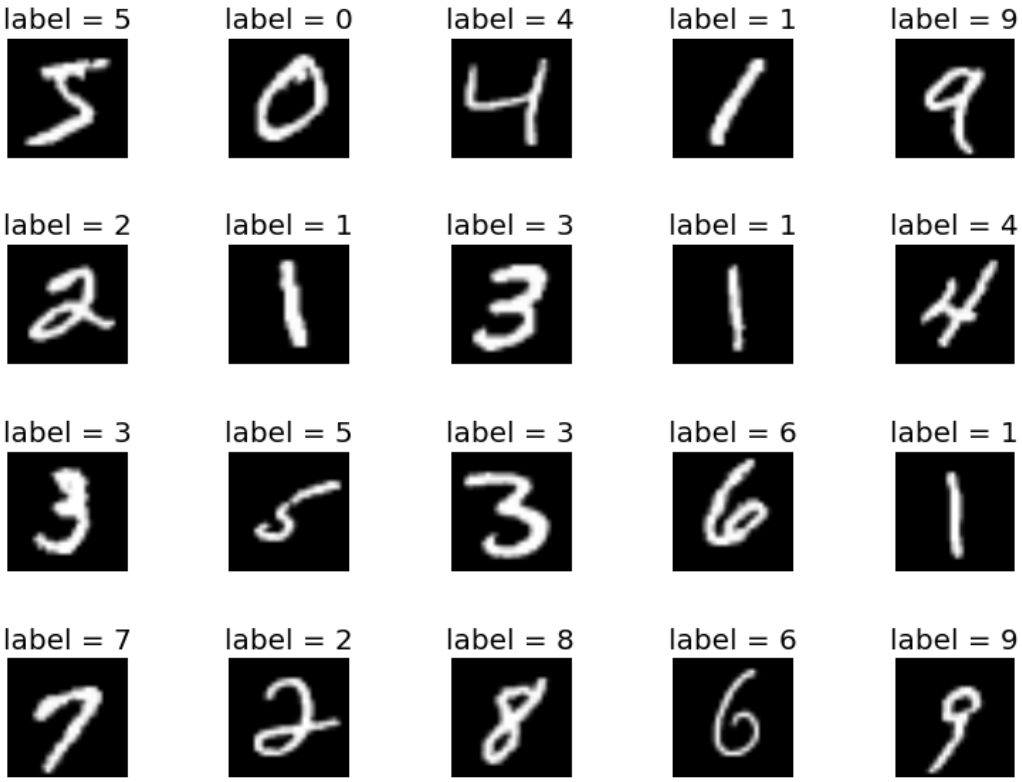

Figure 1: Example training instances from the MNIST database with labels.

Naive Bayesian Learning

In this project, you will implement a naive Bayes classifier for classifying images of handwritten digits. Given an image representing a handwritten digit between 0 and 9, your goal is to output the digit represented by the image. You will train and test this algorithm using the **MNIST database**, one of the most famous machine learning databases[1]. It contains 60,000 training images and their associated **labels** (i.e., the correct digit represented by each image). It also contains 10,000 test images and their associated labels. You will train your naive Bayes classifier on the 60,000 training images (the **training set**) and evaluate its performance on the 10,000 test images

[1] http://yann.lecun.com/exdb/mnist/

(the **test set**). In contrast to Project 2, there will be no validation set. Figure 1 shows a number of examples of MNIST training set images and their associated labels. Correctly identifying images of digits is vitally important in many applications, such as automatically reading the zip codes on pieces of US mail.

Data

The MNIST database is provided for you in the folder mnist_data/, which consists of four files:

1. train-images-idx3-ubyte, which contains 60,000 28×28 grayscale training images, each representing a single handwritten digit.
2. train-labels-idx1-ubyte, which contains the associated 60,000 labels for the training images.
3. t10k-images-idx3-ubyte, which contains 10,000 28×28 grayscale testing images, each representing a single handwritten digit.
4. t10k-labels-idx1-ubyte, which contains the associated 10,000 labels for the testing images.

Files 1 and 2 are the training set. Files 3 and 4 are the test set. Each training and test instance in the MNIST database consists of a 28×28 grayscale image of a handwritten digit and an associated integer label indicating the digit that this image represents (0-9). Each of the $28 \times 28 = 784$ pixels of each of these images is represented by a single 8-bit color channel. Thus, the values each pixel can take on range from 0 (completely black) to 255 ($2^8 - 1$, completely white). To simplify the problem for this project, we are going to **binarize** this data set. Any pixel whose value is lower than 128 will be set to 0 (black) and any pixel whose value is 128 or greater will be set to 255 (white).

Example/skeleton code is provided in the src/ directory. The provided header files mnist_reader_common.hpp, mnist_reader_less.hpp, mnist_reader.hpp, and mnist_utils.hpp implement functions for easily parsing the MNIST database files. main.cpp provides example code for using these functions. You will only need to use two of these functions to solve this problem:

```
std::string MNIST_DATA_DIR = "../mnist_data";
//Read in the data set from the files
 mnist::MNIST_dataset<std::vector, std::vector<uint8_t>, uint8_t> dataset =
 mnist::read_dataset<std::vector, std::vector, uint8_t, uint8_t>(MNIST_DATA_DIR);
 //Binarize the data set (so that pixels have values of either 0 or 1)
 mnist::binarize_dataset(dataset);
```

The first two lines of the above code block (which you can use as-is) load the MNIST database, while the third line binarizes the training and test images, so that each pixel value in each image will be either a 0 (representing black) or 1 (representing white). **Note that white is represented by a 1 when we binarize, not 255!** You can extract the training images and labels as follows:

```
// get training images
std::vector<std::vector<unsigned char>> trainImages = dataset.training_images;
// get training labels
std::vector<unsigned char> trainLabels = dataset.training_labels;
```

The variable $trainImages$ in the code block above is a vector of 60,000 vectors of unsigned chars, each representing a training image. For each $i \in \{0, ..., 59,999\}$, $trainImages[i]$ is a vector of 784 unsigned chars, each representing one of the 784 pixel values of training image i (**linearized in row major order**). The variable $trainLabels$ in the code block above is a vector of 60,0000 unsigned chars. For each $i \in \{0, ..., 59,999\}$, $trainLabels[i]$ is an unsigned char (a single byte) indicating the true digit represented by $trainingImages[i]$. To use these values, they should first be converted to integers (or uint8_t) as follows:

```
//get pixel value j of training image i (0 for black, 1 for white)
int pixelValue = static_cast<int>(trainingImages[i][j])
//get the label (correct digit) of training image i
int label = static_cast<int>(trainingLabels[i]);
```

Note that the variable *label* in the code block above will be an integer in the range [0 - 9], while the variable $pixelValue$ will be either 0 (if the pixel is black) or 1 (if the pixel is white), because we have binarized the data set. The images and the labels in the test set can be extracted analogously and also should be converted to ints or uint8_ts before use:

```
//get the test images
std::vector<std::vector<unsigned char>> testImages = dataset.test_images;
//get the test labels
std::vector<unsigned char> testLabels = dataset.test_labels;
```

Algorithm Framework

We will build a naive Bayes model for this problem by **considering each pixel individually as a binary feature** that takes on the value 0 or false when the pixel is black and 1 or true when the pixel is white. In other words, feature F_j is a random variable representing the color of pixel j, for all $j \in \{0, ..., 783\}$. The class variable, C, will have 10 possible values, one for each possible digit 0-9. From our training set, we will learn the **prior probability** of each class $c \in C$, $P(c)$. We will also learn the **class conditional probabilities** of each feature, given its class. That is, for all $c \in C$, we will learn $P(F_j|c)$, where $P(F_j|c)$ is a distribution over the possible values that F_j can take on (0 or 1, in this case), given that the class (digit) is c. Once we have **learned these probabilities from the training set**, we will **evaluate our model on the test images**. Let T_i represent test image i and let $t_{i,j}$ denote the value (0 or 1) of pixel j in test image i. For each test image i, we will use our model to compute $P(C|F_0 = t_{i,0}, ..., F_{783} = t_{i,783})$, the **posterior probability distribution** over the possible values of the class variable (in this case, the possible digits that the image might represent) given the pixel values of the test image. We will predict that test image T_i represents the digit with the highest posterior probability, that is,

$\hat{c}_i = argmax_c P(C = c | F_0 = t_{i,0}, ..., F_{783} = t_{i,783})$. Let c_i^* represent the correct label (digit) of test image T_i. If $c_i^* = \hat{c}_i$, our model has successfully classified image T_i. The **accuracy** of our model on the test set is:

$$\frac{\sum_{i=0}^{9,999} \mathbb{1}(c_i^* = \hat{c}_i)}{10,000}$$

Note that $\mathbb{1}$ is the indicator function, which returns 1 when $c_i^* = \hat{c}_i$ and 0 otherwise. So the accuracy of our model is simply the ratio of successful classifications to total test images.

Since the intermediate probabilities can be quite small, we use the summation of log probabilities instead of the multiplication of probabilities in order to avoid underflow errors. In other words, instead of calculating $\prod_i P_i$, you will calculate $\sum_i \log P_i$ since $\prod_i P_i = \exp\{\sum_i \log P_i\}$. This implies that you will store $\log P_i$ instead of P_i for all of the following probabilities.

To learn the naive Bayes classifier, you should do all the following steps **only for the training set**.

- Determine the prior probabilities for each class $c \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$:

  $$P(C = c) = \frac{\#\{\text{images of digit } c\}}{\#\{\text{images}\}}$$

  Make sure that: $\sum_{c=0}^{9} P(c) = 1$.

- For each pixel F_j for $j \in \{0, ..., 783\}$ and for each class $c \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, determine $P(F_j = 1 | C = c)$, the probability that pixel F_j is white given that it is an image of digit c:

  $$P(F_j = 1 | C = c) = \frac{\#\{\text{images of digit } c \text{ where pixel } F_j \text{ is white}\}}{\#\{\text{images of digit } c\}}$$

  Note that, if $P(F_j = 1 | C = c) = 0$ (if pixel F_j has never been white in any image of digit c), then, for any image in the test set of digit c where pixel F_j is white, our classifier will predict that the probability that the image is of digit c is 0. To address this issue, we can pretend that we have observed every outcome once more than we actually did (called Laplace smoothing):

  $$P_L(F_j = 1 | C = c) = \frac{\#\{\text{images of digit } c \text{ where pixel } F_j \text{ is white}\} + 1}{\#\{\text{images of digit } c\} + 2}$$

  Note that $P(F_j = 0 | C = c) = 1 - P(F_j = 1 | C = c)$ since each feature is binary.

Once you have trained your naive Bayes classifier using the training set, you need to evaluate its performance on the **test set**. For a test image T_i and pixel F_j, let $t_{i,j}$ denote the value of pixel j in test image T_i. Note that $t_{i,j}$ will be either 0 or 1 because we have binarized the test set as well as the training set.

For each test image T_i, compute the probability that it belongs to each class $c \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$:

$$P(C = c|F_0 = t_{i,0}, ..., F_{783} = t_{i,783}) = \frac{P(F_0 = t_{i,0}, ..., F_{783} = t_{i,783}|C = c) \cdot P(C = c)}{P(F_0 = t_{i,0}, ..., F_{783} = t_{i,783})}$$

$$= \frac{(\prod_{j \in \{0,...,783\}} P_L(F_j = t_{i,j}|C = c)) \cdot P(C = c)}{P(F_0 = t_{i,0}, ..., F_{783} = t_{i,783})}$$

Where the second step is due to the Naive Bayes assumption.

The test image T_i is classified as belonging to the class c with the highest posterior probability $\hat{c}_i = argmax_c P(C = c|F_0 = t_{i,0}, ..., F_{783} = t_{i,783})$. Notice that the term $P(F_0 = t_{i,0}, ..., F_{783} = t_{i,783})$ can be dropped from the calculations without changing the resulting classification. Since we are using Laplace smoothing, so we are using $P_L(F_j = t_{i,j}|C = c)$ rather than $P(F_j = t_{i,j}|C = c)$. Furthermore, remember that, in order to avoid underflows, instead of multiplying probabilities, we are adding their logarithms. Therefore, we classify image T_i as belonging to the class $\hat{c}_i$ that maximizes the following expression:

$$argmax_c(\sum_{j \in \{0,...,783\}} \log P_L(F_j = t_{i,j}|C = c)) + \log P(C = c)$$

The reference implementation performed the maximization using the std::max_element() function defined in the $< algorithm >$ header file, if you wish to be fully consistent with it.

Evaluation

Working with images affords us unique opportunities to visualize the parameters we have learned for our naive Bayes classifier. For the first part of this evaluation, you will use the provided Bitmap class (also in the src/ directory and defined in bitmap.cpp and bitmap.hpp) to output images representing the class conditional densities associated with each class (i.e., $P(F_j = 1|C = c)$). Use the following code to visualize the Naive Bayes model you have learned:

```
int numLabels = 10;
int numFeatures = 784;
for (int c=0; c<numLabels; c++) {
    std::vector<unsigned char> classFs(numFeatures);
    for (int f=0; f<numFeatures; f++) {
     //TODO: get probability of pixel f being white given class c
     p = probability that pixel f is white given digit c
       uint8_t v = 255*p;
       classFs[f] = (unsigned char)v;
    }
    std::stringstream ss;
```

```
        ss << "../output/digit" <<c<<".bmp";
        Bitmap::writeBitmap(classFs, 28, 28, ss.str(), false);
    }
```

Note that to finish the code block above you must assign to the variable p the probability that the pixel at location f is white given that the class is c after learning those probabilities from the training set. The remainder of the code can be executed as is. This will write 10 bitmap files to the output directory digit$i$.bmp for $i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The results should look like Figure 2.

Figure 2: Visualization of the class conditional densities of each pixel for each class $c \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Your program should output two additional files:

- **network.txt** This file should list the values of all conditional probabilities parameterizing the network. The first 784 lines should be $P(F_j = 1|C = 0)$. The next 784 lines should be $P(F_j = 1|C = 1)$. The final 10 lines should be the prior probabilities of each class, in order from 0 to 9.
- **classification-summary.txt** This file should contain a 10×10 matrix of integers. The integer in row r and column c should be the number of images in the test set of digit r which our model predicted was an image of digit c. The final line of this file should be the final accuracy of your Naive Bayes model (number correctly classified/10,000) on the test set of 10,000 images. For comparison, the reference implementation achieves a test accuracy of **84.43%**.

Finally, we note that using Naive Bayes to solve this problem does not represent the current state of the art. For instance, a relatively simple convolutional neural network can achieve an accuracy of about 99.2%. A committee of 5 convolutional neural networks can achieve an accuracy of approximately 99.8%. The Python TensorFlow library has a very informative tutorial on building such a model for the same digit recognition task at https://www.tensorflow.org/get_started/mnist/pros. Though it is not part of this project, we encourage you to think about why such models might perform better than the Naive Bayes model we have built in this project.

Submission

You need to submit your solution on Blackboard as a zip file that contains your source code and a makefile that produces an executable called proj3. When we run your executable, it should process the training and test sets and output the bitmap files and the two text files mentioned in the Evaluation section. **Please do not include the mnist_data folder in your submission, or it will take a very long time to upload to Blackboard**.