MNIST Digit Classification

Derek Wayne

08 April 2019

Abstract

In this note, we explore the differences in capabilities and performance of generative and discriminative models fit to the MNIST dataset of handwritten digits. There are some readily apparent advantages to using a probabilistic classifier such as naive Bayes; generating new data and being able to handle missing data, for example. However, the naive Bayes independence assumption is often considered too strong and results in a less than optimal misclassification rate. Using a predictive model can improve performance without needing to account for correlated features.

Naive Bayes

Each datapoint is regarded as a 28 by 28 pixel greyscale image of handwritten digits ($\{0, 1, \dots, 9\}$), and a classification label indicating the true digit. Thus we can state the problem as follows: Given the image of a written digit, what is the true digit meant to be? That is, given datapoint $\mathbf{x}$, what is its true label? Our first approach will be Bayesian. We will also binzarize the pixels values, using > 0.5 as the cut-off.

Given input space $\chi = \{0, 1\}^{784}$ and target space $\mathcal{Y} = \{0, 1, \dots, 9\}$ let $c \in \mathcal{Y}$ where the distribution of target c is given by $p(c|\pi) = \pi_c$. Hence for a given image $\mathbf{x}$, and class c, the joint density is given by

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

For a given class/label c and a pixel d. We model the pixel state (white or black) with the Bernoulli likelihood:

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

it follows that $\theta_{cd} = P\{x_d = 1 \,|\, c, \theta_{cd}\}$ for a single example. Let $\mathbf{y}$ be the $N = 60{,}000$-dimensional vector of predictions for the dataset. Then we may estimate the conditional pixel means θ_{cd} by maximum likelihood estimation as follows,

$$\widehat{\theta}_{cd} := arg \max_{\theta_{cd}} \prod_{n=1}^{N} \pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d^{(n)}} (1 - \theta_{cd})^{1-x_d^{(n)}}$$

$$= arg \max_{\theta_{cd}} \sum_{n=1}^{N} \left(log\, \pi_c + \sum_{d=1}^{784} x_d^{(n)} log\, \theta_{cd}^{(n)} + (1 - x_d^{(n)}) log\, (1 - \theta_{cd}^{(n)}) \right)$$

which is the solution to the likelihood equation:

$$\sum_{n=1}^{N} I\{y^{(n)} = c\} \left(x_d^{(n)}(1 - \theta_{cd}) - (1 - x_d^{(n)})\theta_{cd} \right) = 0$$

and so,

$$\widehat{\theta}_{cd,MLE} = \frac{\sum_{n=1}^{N} I\{y^{(n)} = c\} x_d^{(n)}}{\sum_{n=1}^{N} I\{y^{(n)} = c\}}$$

which is to be interpreted as the ratio of the number of examples with the dth pixel activated in class c to the total number of examples of class c, which is a natural result for the MLE. However, this estimator can be problematic if there are no examples of a particular class within the dataset (thus we have 0 in the denominator of our estimator), or the pixel value is 1 in every example in dataset (then the estimator is equal to 1).

Consider instead the maximum aposteriori estimator with a $Beta(\alpha, \beta)$ prior, given by,

$$\widehat{\theta}_{cd,MAP} := arg \max_{\theta_{cd}} P\{\theta_{cd} \,|\, x_d, c\}$$

$$= arg \max_{\theta_{cd}} \prod_{n=1}^{N} \pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d^{(n)}} (1-\theta_{cd})^{1-x_d^{(n)}} \times Beta(\alpha, \beta)$$

$$= arg \max_{\theta_{cd}} \left(\theta_{cd}^{\sum_{n=1}^{N} I\{y^{(n)}=c\} x_d^{(n)} + (\alpha-1)} (1-\theta_{cd})^{\sum_{n=1}^{N} I\{y^{(n)}=c\}(1-x_d^{(n)}) + (\beta-1)} \right)$$

where the expression inside parenthesis on the last line may be recognized as $Beta(\sum_{n=1}^{N} I\{y^{(n)} = c\} x_d^{(n)} + \alpha\,, \sum_{n=1}^{N} I\{y^{(n)} = c\}(1 - x_d^{(n)}) + \beta)$ which for $\alpha, \beta > 1$ (hence unimodal), has mode:

$$\frac{\sum_{n=1}^{N} I\{y^{(n)} = c\} x_d^{(n)} + \alpha - 1}{\sum_{n=1}^{N} I\{y^{(n)} = c\} + \alpha + \beta - 2}$$

In particular if we let $\alpha = \beta = 2$ (symmetric about 1/2) we have,

$$\widehat{\theta}_{cd,MAP} = \frac{\sum_{n=1}^{N} I\{y^{(n)} = c\} x_d^{(n)} + 1}{\sum_{n=1}^{N} I\{y^{(n)} = c\} + 2}$$

This is known as "Laplacian smoothing". Then the 784×10 matrix $\widehat{\theta}_{MAP}$ will have entry c, d given as derived above.

Next we will load in the MNIST dataset from the loader provided in [http://yann.lecun.com/exdb/mnist/].

```r
source('loadMNIST.R')
load_mnist()
train$x <- ifelse(train$x > 0.5, 1, 0) # binarize pixels
test$x <- ifelse(test$x > 0.5, 1, 0)
```

Utilizing the derived MAP estimates for the pixel means we fit θ to the training set:

```r
pixel_means <- function(data) {
  m <- matrix(0, nrow=784, ncol=10)
  for (i in 0:9) {
    class_set <- data$x[data$y == i, ]
    theta.c <- (colSums(class_set) + 1) / (nrow(class_set) + 2)
    m[ ,i+1] <- theta.c
  }
  m
}
theta.hat <- pixel_means(train)
```

Above are the learned weights for $\widehat{\theta}_{MAP}$.

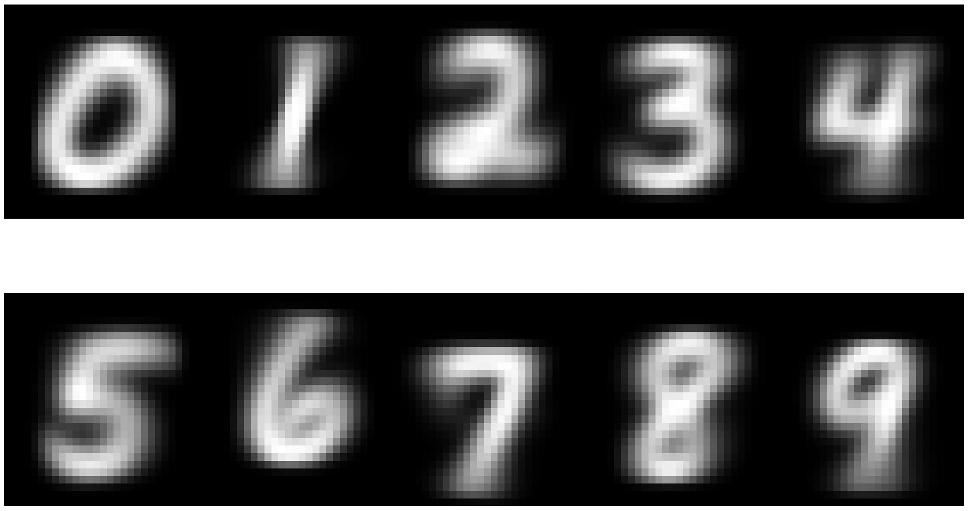

Figure 1: Condition Pixel Means

For a single training image we have the log-likelihood as,

$$log\, p(c\,|\,\mathbf{x}, \theta, \pi) = log\left\{\frac{\pi_c\, p(\mathbf{x}\,|\,c, \theta, \pi)}{\sum_{c'=0}^{9} \pi_{c'}\, p(\mathbf{x}\,|\,c', \theta, \pi)}\right\}$$

$$\propto log\left(\pi_c\, p(\mathbf{x}\,|\,c, \theta, \pi)\right)$$

$$= log\left\{\pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d}(1-\theta_{cd})^{1-x_d}\right\}$$

$$= log\pi_c + \sum_{d=1}^{784}\left(x_d log\,\theta_{cd} + (1-x_d) log(1-\theta_{cd})\right)$$

Let $\pi_c = \frac{1}{10}$ and let us find the average log-likelihood per datapoint ($\frac{1}{N}\sum_{i=1}^{N} log\, p(c_i\,|\,x_i, \theta, \pi)$),

```r
loglik <- function(datapoint, prior, theta) {
  image <- datapoint$x; class <- datapoint$y
  r <- log(prior) + sum(image * log(theta[,class+1]) +
                          (1-image) * log(1-theta[,class+1]))
  r
}

ll <- c()
for (d in 1:nrow(train$x)) {
  datapoint <- list(x=train$x[d,], y=train$y[d])
  ll <- c(ll, loglik(datapoint, 1/10, theta.hat))
}
mean(ll)
```

```
## [1] -192.1601
```

Thus we have found that $\frac{1}{N}\sum_{i=1}^{N} log\, p(c_i \,|\, x_i, \theta, \pi)$ = -192.1600733. Next we will compute the accuracy; defined as the fraction of examples where the true class $t = arg\ \max_c p(c \,|\, \mathbf{x}, \theta, \pi)$.

```r
bayes_estimate <- function(image) {
  logliks <- c()
  for (i in 0:9) {
    datapoint <- list(x=image, y=i)
    logliks <- c(logliks, loglik(datapoint, 1/10, theta.hat))
  }
  which.max(logliks) - 1
}
```

```r
train.estimates <- numeric(nrow(train$x))
test.estimates <- numeric(nrow(test$x))
for (i in 1:nrow(train$x)) {
  train.estimates[i] <- bayes_estimate(train$x[i,])
}
for (i in 1:nrow(test$x)) {
  test.estimates[i] <- bayes_estimate(test$x[i,])
}
accuracy.train <- mean(train$y == train.estimates)
accuracy.test <- mean(test$y == test.estimates)
accuracy.train
```

```
## [1] 0.8313333
```

```r
accuracy.test
```

```
## [1] 0.8415
```

The Naive Bayes approach gives a training data classification accuracy of 0.8313333; and a test accuracy of 0.8415.

Again, the Naive Bayes assumption is that within each class, distinct pixel values are independent (i.e. $x_i \perp x_j \,|\, c,\ i, j \in \{1, \ldots, 784\},\ i \neq j$). As such we have, $p(\mathbf{x} \,|\, c, \theta_c) = \prod_{d=1}^{784} \theta_{cd}^{x_d}(1 - \theta_{cd})^{1-x_d}$. However, if we marginalize over c, and instead consider an assumption of unconditional independence (i.e. $x_i \perp x_j,\ i \neq j$) then things break down. Observing pixels in one region of the image would lead one to expect pixels in the same region of another image of the same class, suggesting a strong correlation between elements of the feature space.

To demonstrate the effects of our assumptions we randomly sample from the marginal distribution $p(\mathbf{x} \,|\, \theta, \pi)$, using our trained model.

```r
samples <- matrix(0, nrow=784, ncol=10)
for (i in 1:10) {
  c <- sample(0:9, 1)
  theta.c <- theta.hat[,c+1]
  samp <- numeric(784)
  for (j in 1:784) {
    samp[j] <- sample(c(0,1), 1, prob = c(1-theta.c[j], theta.c[j]))
  }
  samples[,i] <- samp
}
```

It appears as if the Naive Bayes model may be too simplistic. The conditional independence assumption could be too strong.

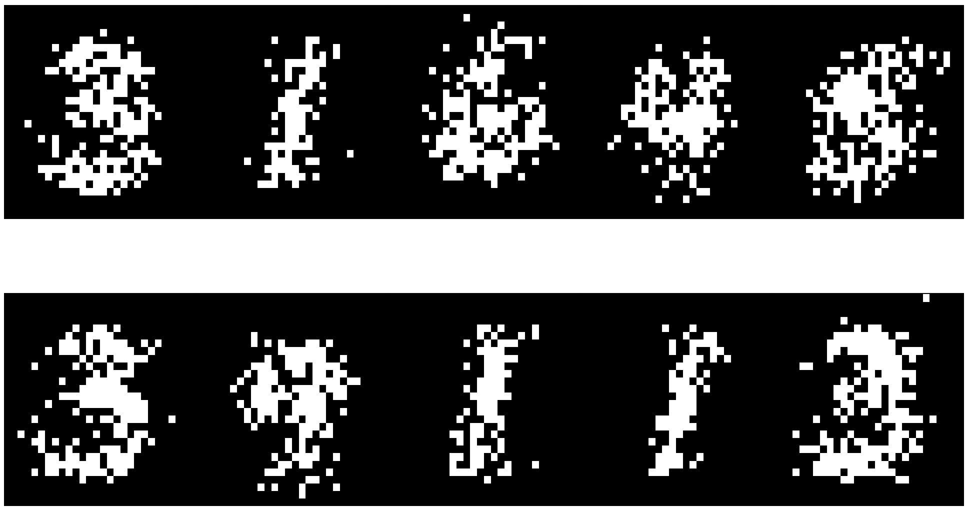

Figure 2: Random Samples From Model

Here we consider how the Naive Bayes models handles missing data. Say we are given the top half of an image and wish to find the marginal distribution of a single pixel in the bottom half. Then let $B = \{392, \ldots, 784\}$

$$p(x_{i \in B} \,|\, \mathbf{x}_{top}\,, \theta, \pi) = \sum_{c} p(x_{i \in B} \,|c\,, \mathbf{x}_{top}\,, \theta, \pi) \frac{p(\mathbf{x}_{top} \,|\, c, \theta, \pi) p(c \,|\, \pi)}{\sum_{c'} p(\mathbf{x}_{top} \,|\, c, \theta, \pi) p(c \,|\, \pi)}$$

$$= \sum_{c} p(x_{i \in B} \,|c\,, \theta, \pi) \frac{p(\mathbf{x}_{top} \,|\, c, \theta, \pi) p(c \,|\, \pi)}{\sum_{c'} p(\mathbf{x}_{top} \,|\, c, \theta, \pi) p(c \,|\, \pi)}$$

futhermore, if we continue to assume that $p(c \,|\, \pi)$ is the same for all c, then the expression simplifies further to $S^{-1} \sum_c p(x_{i \in B} \,|c\,, \theta, \pi) p(\mathbf{x}_{top} \,|\, c, \theta, \pi)$ where S is the sum in the denomitor of the expression above. From the results derived above, we will plot the top half the image concatenated with the marginal distribution over each pixel in the bottom half.

```
marginal_means <- function(image) {
  marginal <- c()
  for (i in 393:784) {
    denom <- 0
    tmp <- numeric(10)
    for (j in 1:10) {
      bottom_pix <- dbinom(image[i], 1, theta.hat[i, j])
      top_probs <- prod(dbinom(image[1:392], 1, theta.hat[1:392,j]))
      denom <- denom + top_probs
      tmp[j] <- bottom_pix * top_probs
    }
    marginal <- c(marginal, 1-(sum(tmp) / denom))
  }
  new_image <- c(image[1:392], marginal)
  new_image
  }
```

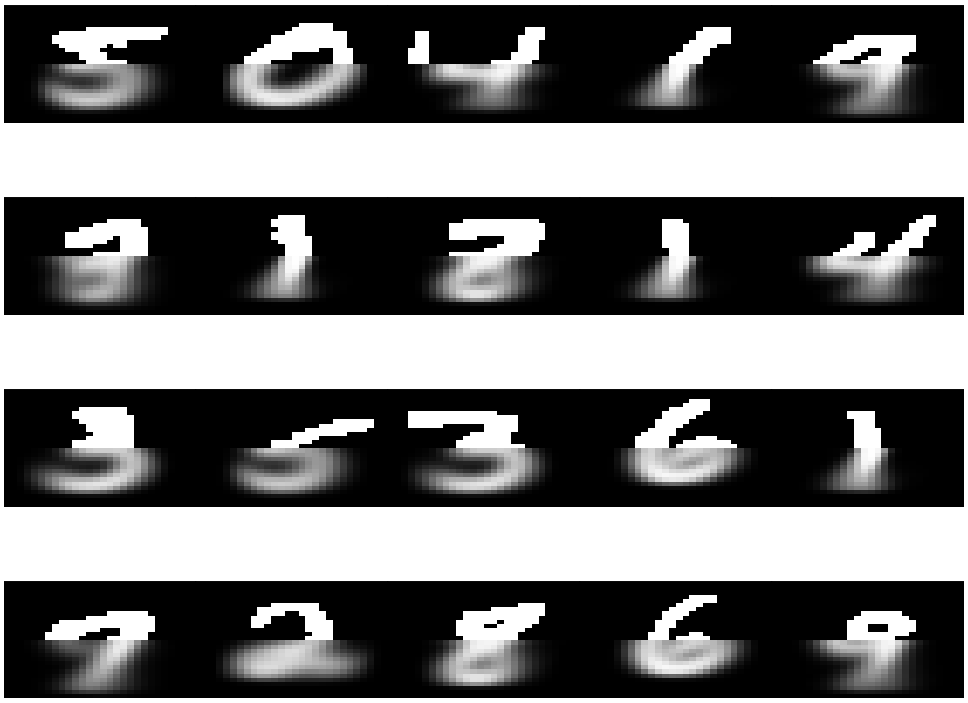

Figure 3: Marginal distribution of a single pixel given top half

Logistic Regression

In this section we will compare our previous results with a multiclass logistic regression model.

$$p(c \,|\, \mathbf{x}, \mathbf{w}) = \frac{exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c\prime=0}^{9} exp(\mathbf{w}_{c\prime}^T \mathbf{x})}$$

which is the softmax function, which we denote by S_c.The model has $784 \times 10 = 7840$ wieght parameters (thus we supress a bias parameter here). We can express the log-softmax as $\mathbf{w}_c^T \mathbf{x} - log\Bigg(\sum_{c\prime=0}^{9} exp(\mathbf{w}_{c\prime}^T \mathbf{x})\Bigg)$. For the batch of $N = 60,000$ training examples the predictive log-likelihood is then given by $log\mathcal{L}(\mathbf{w}) = \sum_{n=1}^{N} \Bigg(\mathbf{w}_c^T \mathbf{x} - log\Bigg(\sum_{c\prime=0}^{9} exp(\mathbf{w}_{c\prime}^T \mathbf{x})\Bigg)\Bigg)$. To arrive at the gradient, we compute the partial derivatives with respect to each paramter:

$$\frac{\partial log\mathcal{L}(\mathbf{w})}{\partial w_c^{(d)}} = \frac{\partial}{\partial w_c^{(d)}} \sum_{n=1}^{N} w_{c'}^T x_n - \frac{\partial}{\partial w_c^{(d)}} \sum_{n=1}^{N} log \sum_{c'=0}^{9} exp(w_{c'}^T x_n)$$

$$= \sum_{n=1}^{N} I\{y_n = c\} x_n^{(d)} - \sum_{n=1}^{N} \frac{\sum_{c'=0}^{9} exp(w_{c'}^T x_n) I\{y_n = c\} x_n^{(d)}}{\sum_{c'=0}^{9} exp(w_{y_n}^T x_n)}$$

$$= \sum_{n=1}^{N} \Bigg(I\{y_n = c\} x_n^{(d)} - S_c x_n^{(d)} \Bigg)$$

for $c = 0, 2, \ldots, 9$ and $d = 1, \ldots, 784$. However, there is a more intuitive interpretation. If we consider the vector of probabilities (of being in each of the 10 classes) for image i, $\widehat{\mathbf{y}}_i = [\hat{y}_i^{(1)}, \ldots, \hat{y}_i^{(10)}]$ given by the output of the softmax activation function, then we may arrive at the useful representation of a single observation:

$$P(\widehat{\mathbf{y}}_i) = \prod_{j=1}^{10} \hat{y}_j^{y_j}$$

$$= (1) \times (1) \times \cdots \times \hat{y}_k^1 \times \cdots \times (1)$$

Thus, as we have $N = 60000$ examples/training images, we find the log-likelihood to be

$$\sum_{n=1}^{N} logP(\widehat{\mathbf{y}}_1, \ldots, \widehat{\mathbf{y}}_N) = \sum_{n=1}^{N} \sum_{j=1}^{10} y_j log\hat{y}_j$$

Below we will implement a gradient descent procedure to maximize our predictive log-likelihood. However, this is equivalent to minimizing the negative log-likelihood: $-\sum_{n=1}^{N} \sum_{j=1}^{10} y_j log\hat{y}_j$ which is defined to be the cross entropy. Thus we shall use this as our cost function. For efficiency and ease of computation we make use of the tensorflow framework.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

train_epochs = 100
# tf Graph Input
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))

model = tf.nn.softmax(tf.matmul(x, W))
cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(model), reduction_indices=1))

eta = 0.5
grad_W = tf.gradients(xs = W, ys = cost)
update = W.assign(W - tf.scalar_mul(eta, grad_W[0]))
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  ll_sum = 0
  for epoch in range(train_epochs):
    u, c = sess.run([update, cost], feed_dict = {x: r.train_images, y: r.train_labels})
    ll_sum += c
  correct_prediction = tf.equal(tf.argmax(model, 1), tf.argmax(y, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  sess.run(accuracy, feed_dict={x: r.train_images, y: r.train_labels})
  sess.run(accuracy, feed_dict={x: r.test_images, y: r.test_labels})
  print(ll_sum/train_epochs)
```

```
## 0.8933091
## 0.8997
## 0.5649890029430389
```

We have achieved an accuracy of about 0.89 on the training data and about 0.90 on the test data. The average log-likelihood was found to be about -0.565. Compared to Naive Bayes this is an improvement. The

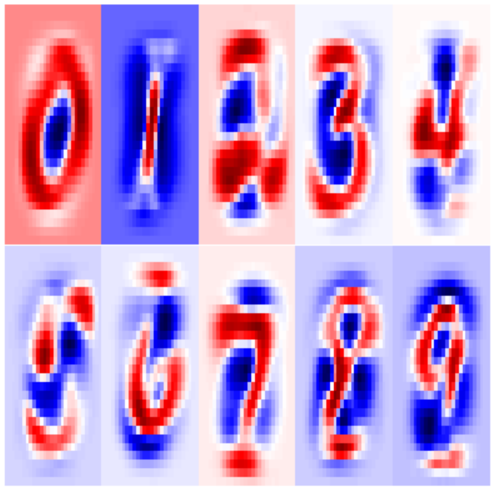

Figure 4: Learned Weights

gradient descent method could be further optimized by using mini-batch learning. The comparison to Naive Bayes loosely demonstrates the trade-offs between model complexity and accuracy.