pROJECT paRT 1:  
dENSITY eSTIMATION AND cLASSIFICATION

September 21, 2019

## Project Background and Description

|  |  |
| --- | --- |
|  | In this project we were required to first perform a parameter estimation for a given dataset (which is a subset from the MNIST dataset). Then we must use these estimated distributions for doing the Naïve Bayes and the Logistic Regression classification. We also must report the classification accuracy for both the models. This report includes the dataset overview, formulas used for the estimates, explanations for how the distributions are used in classifying the testing samples. |

## MNIST Dataset

The MNIST dataset contains 70,000 images of handwritten digits, divided into 60,000 training images and 10,000 testing images.  It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

The data is stored in a very simple file format designed for storing vectors and multidimensional matrices.

The digit images in the MNIST set were originally selected and experimented with by Chris Burges and Corinna Cortes using bounding-box normalization and centering.



## Training and Test Sets: Splitting Data

We have only got a subset of the MNIST dataset. This subset contains images for digit “7” and digit “8”. The splitting of the dataset into training and testing sets, was already been done for us. we have the following statistics for the given dataset:

* Number of samples in the training set:  "7": 6265;"8": 5851.
* Number of samples in the testing set: "7": 1028; “8": 974

## Parameter Estimation

In this project we were required to the following features for each image:

* The average of all pixel values in the image.
* The standard deviation of all pixel values in the image.

For calculating the mean and standard deviation, I used the following two functions from the numpy library available in python:

* **numpy.mean(***a***,***axis=None***,***dtype=None***,***out=None***,***keepdims=False***)**
  + Compute the arithmetic mean along the specified axis.
  + Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs.
  + I made use of the axis parameter to calculate the mean over the series of rows (i.e. images, which I was required to do). In addition to this I took a mean each pixel wise as well.
* **numpy.std(***a***,***axis=None***,***dtype=None***,***out=None***,***ddof=0***,***keepdims=False***)**
  + Compute the standard deviation along the specified axis.
  + Returns the standard deviation, a measure of the spread of a distribution, of the array elements. The standard deviation is computed for the flattened array by default, otherwise over the specified axis.

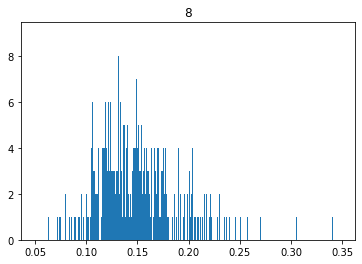
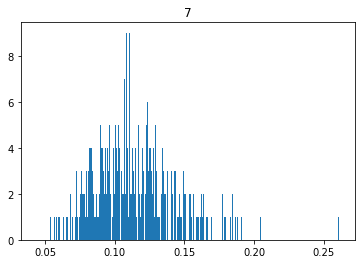


Figure 2 Histogram graph for the means calculated row-wise for number '7', i.e. for every image taking the mean value.

Figure 1 Histogram graph for the means calculated row-wise for number 8, i.e. for every image taking the mean value.

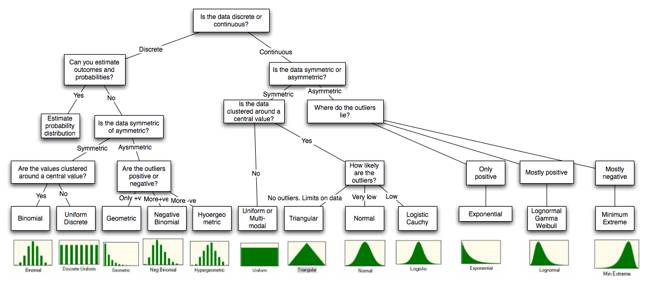


Figure 3 Distributional choice

For my further calculations, I will assume that the graph follows a normal distribution, which is a fair estimate given the histogram graphs we saw in Figure 1 and Figure 2. Why is taking normal graph so important and why it is so liked by the statisticians around the world? Well that is because of the following two features:

* Normal distribution is characterized by just two parameters which are the mean and the standard deviation. This factor reduces are various number of calculations required in other type of graphs.
* The probability of any value can be estimated by calculating how many standard deviations does the value differ from the mean. In other words, we take advantage of the symmetry in the graph.

By taking the mean value for each image we are taking the overall intensity of the image to be a feature to identify a given number. For more explanation let us assume the digits “1” and “8”, I am assuming these two because the explanation will become way more intuitive with this. Generally, the overall intensity of the image with “1” should be less than “8”, given all the curves with 8 and 1 can be represented as a straight line. So, basically for our calculations we are counting on that. Another crude approach to our problem can be taking all the pixels individually as features. There are 786(28x28) pixel in any image which basically means we are taking 786 features for each image. This might sound to be absurd to begin with but with makes sense with the intuition of Naïve Bayes given one small assumption. The assumption I took while calculating this was to not to calculate the probability density if the standard deviation is 0. This might not actually reduce our feature space but will reduce the calculations for making an estimate for a label. The features learnt with this methodology can be plotted as follows:

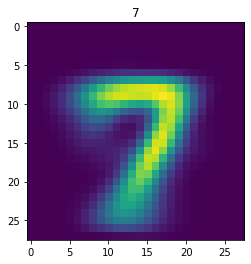
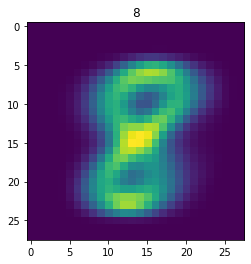


Figure 4. These two images show the mean value of every pixel learnt by Naive Bayes model when taking each pixel as a different feature. Therefore, this is mean of each pixel in the given training set.

## Naïve Bayes

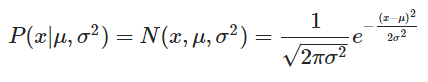
The learning in Naïve Bayes is straightforward. One just has to calculate a bunch of probabilities for a given dataset, which are as follows:

* Class Probabilities: The probability of each class or label in the dataset
* Conditional Probabilities: The conditional probabilities of each input given each label value.

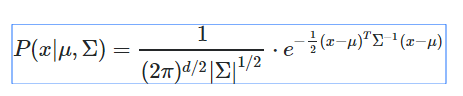
Naïve Bayes is extended to real valued attributes by assuming the distribution to be a Gaussian(also called normal distribution) which is a valid assumption in our case as we discussed earlier in the last section. Learning in gaussian Naïve Bayes can be reduced to two steps:

* Calculating mean value for each input variable:
  + mean(x) = 1/n \* sum(x)
  + here, n is the number of instances and x are the values for an input variable in the training data.
* Calculating the standard deviation:
  + standard deviation(x) = sqrt(1/n \* sum(xi-mean(x)^2 ))

Predictions in Naïve Bayes are made by estimating the probabilities of the new values of x using Probability Density Function (PDF). For a Gaussian distribution the PDF in naïve Bayes can be given as



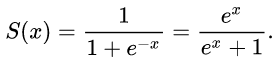
this function is only valid if we have one dimension if we have multiple dimensions lets say d dimensions then the equation becomes:



where μ is the mean and Σ is a d x d covariance matrix. In my code I am manually feeding these equations to predict the data.

## Logistic Regression

To generate probabilities, logistic regression uses a special function which generates values between 0 and 1 for all the input values. This function is called as logistic function. This is also known as sigmoid function



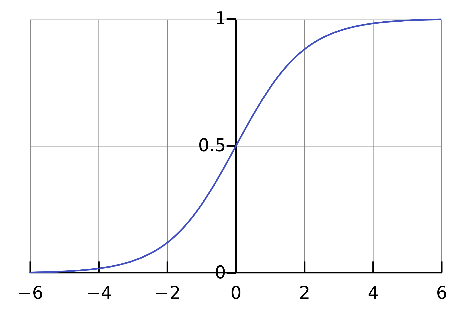


Figure 5 The sigmoid function or logistic function, this has a characteristic "S" shaped curve

We start with a random set of weights or maybe zero weights and iteratively try to estimate the weightage of each feature. The iterations are defined by three major things which are:

* Learning Rate
* Gradient
* Upper Limit to the number of iterations

The learning in logistic regression is basically to find the right weights for the feature given a learning rate and a cap to the number of iterations. The gradient is the small step that we want to take in the real valued knowledge space to reach a particular value. We can have two typed of gradients ascent and descent. Which does not make much of a difference except for the direction in which we move along the tangent. The learning rate our control over the gradient we can make an estimator jump by a multiple of the gradient in order to reduce the number of calculations. This jump is defined by the learning rate. The ultimate aim of a gradient is to minimize the gradient. The gradient is calculated as follows:

h = self.\_\_sigmoid(z)

gradient = np.dot(X.T, (h - y)) / y.size

and the weights are calculated as follows:

self.weights -= self.lr \* gradient

For the predictions, by calling the sigmoid function we get the probabilities that help us identify which label does a given value belong to. This is done by defining a threshold value. The values in logistic function are bound in 0 and 1, this threshold value acts as a horizontal identifier below which is one class and above which is the other class.

## Resources

* <http://yann.lecun.com/exdb/mnist/>
* <http://lagrange.univ-lyon1.fr/docs/numpy/1.11.0/reference/generated/numpy.mean.html>
* <http://lagrange.univ-lyon1.fr/docs/numpy/1.11.0/reference/generated/numpy.std.html>
* <http://people.stern.nyu.edu/adamodar/New_Home_Page/StatFile/statdistns.htm>
* <https://stats.stackexchange.com/questions/8419/bayesian-classifier-with-multivariate-normal-densities>
* <https://en.wikipedia.org/wiki/Sigmoid_function>