

# Bernoulli Classification Model Report

<https://github.com/eunhong0925/gks-4008.git>

20221009 GKS Kim Eunhong

## 01. Theory and Method

The goal of this project is to create a classification model for the MNIST handwritten dataset. The MNIST dataset is a collection of 28x28 pixel grayscale images of handwritten digits ranging from 0 to 9. For this task, Bernoulli Naive Bayes approach is used for doing this classification task. It bases on Bayes' theorem, which uses likelihood and prior to derive the posterior probability.

$$Posterior = \frac{Likelihood \times Prior}{Evidence} \quad P(A_k|B) = \frac{P(B|A_k)P(A_k)}{P(B|A_1)P(A_1) + \dots + P(B|A_n)P(A_n)}$$

It supposes that each pixel is independent and each iamge has binaray data. Additionally, log probabilities is used in the calculation process to reduce errors. Specifically, aimed to implement the probability density function as follows.

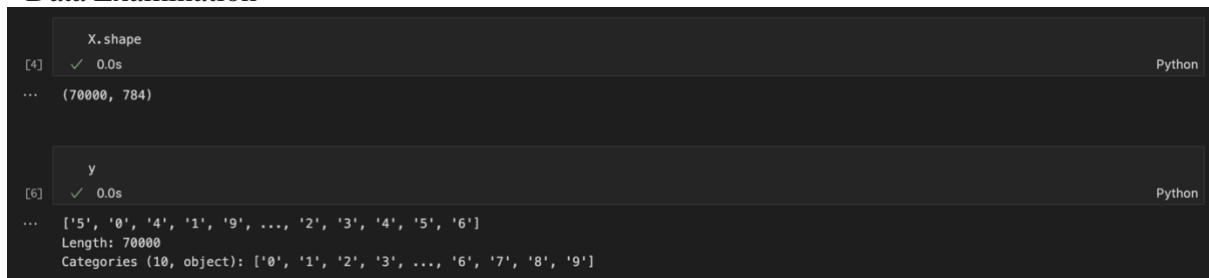
$$\hat{y} = \operatorname{argmax}_y \log P_y[y] + \sum_{i=1}^d \left[ t_i \log P_{xy}[x_i, y] + (1 - t_i) \log(1 - P_{xy}[x_i, y]) \right]. \quad (22.9.7)$$

Specifically, it follows a structure where it calculates the log probabilities (prior probabilities) for each of the ten classes and computes the feature probabilities (conditional probabilities) for the data in each class image. These calculations are based on the 768-dimensional data, where each pixel should be flattened for probability Through this process, it obtains the log posterior probabilities using this approach.

## 02. Implementation

This section will explain the essential steps for understanding and solving the problem and then provide detailed steps thereafter.

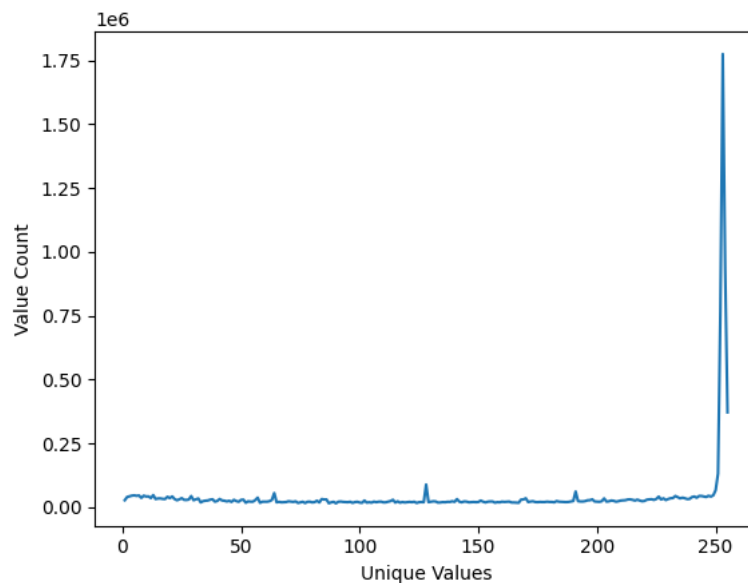
### - Data Examination



```
X.shape
[4] ✓ 0.0s Python
... (70000, 784)

y
[6] ✓ 0.0s Python
... ['5', '0', '4', '1', '9', ..., '2', '3', '4', '5', '6']
Length: 70000
Categories (10, object): ['0', '1', '2', '3', ..., '6', '7', '8', '9']
```

The structure of the MNIST dataset was examined. It contains 70,000 images, each of dimension 784. The labels, represented by 'y,' consist of 10 categories ranging from 0 to 9.



Each data point in the dataset predominantly consists of values near 0 or 256, regarding it essentially binary data. Therefore, the decision was made to use the Bernoulli distribution for modeling.

### - Classification Bernoulli Model

```
#6. define a class
class BernoulliNBWithLog:
    def fit(self, X, y):
        self.classes = np.unique(y)

        # Calculate and store the log of class priors - the likelihood of each class occurring
        self.class_priors = np.array([np.log(np.mean(y == c)) for c in self.classes])

        # Calculate the likelihood of a set of features (the feature vector) given a class
        self.feature_probs = []
        for c in self.classes:
            feature_prob = (X[y == c].sum(axis=0) + 1) / (np.sum(y == c)) # Smoothing
            # Calculate the log of the complement probabilities (1-p) to avoid underflow
            complement_prob = 1 - feature_prob
            self.feature_probs.append((np.log(complement_prob), np.log(feature_prob)))
        self.feature_probs = np.array(self.feature_probs)

    def predict(self, X):
        log_likelihoods = np.zeros((X.shape[0], len(self.classes)))

        for i, c in enumerate(self.classes):
            log_likelihoods[:, i] = np.sum(X * self.feature_probs[i][1] + (1 - X) * self.feature_probs[i][0], axis=1)

        log_posteriors = log_likelihoods + self.class_priors

        predicted_class = self.classes[np.argmax(log_posteriors, axis=1)]
        return predicted_class
```

A Bernoulli distribution classification model was defined. It is primarily structured around two functions, 'fit' and 'predict.' It computes the prior probability and likelihoods to calculate posterior probabilities, ultimately returning the class with the highest probability.

### - Model Accuracy Measurement

```
#8. Evaluate the model
accuracies = []
num_iterations = 100

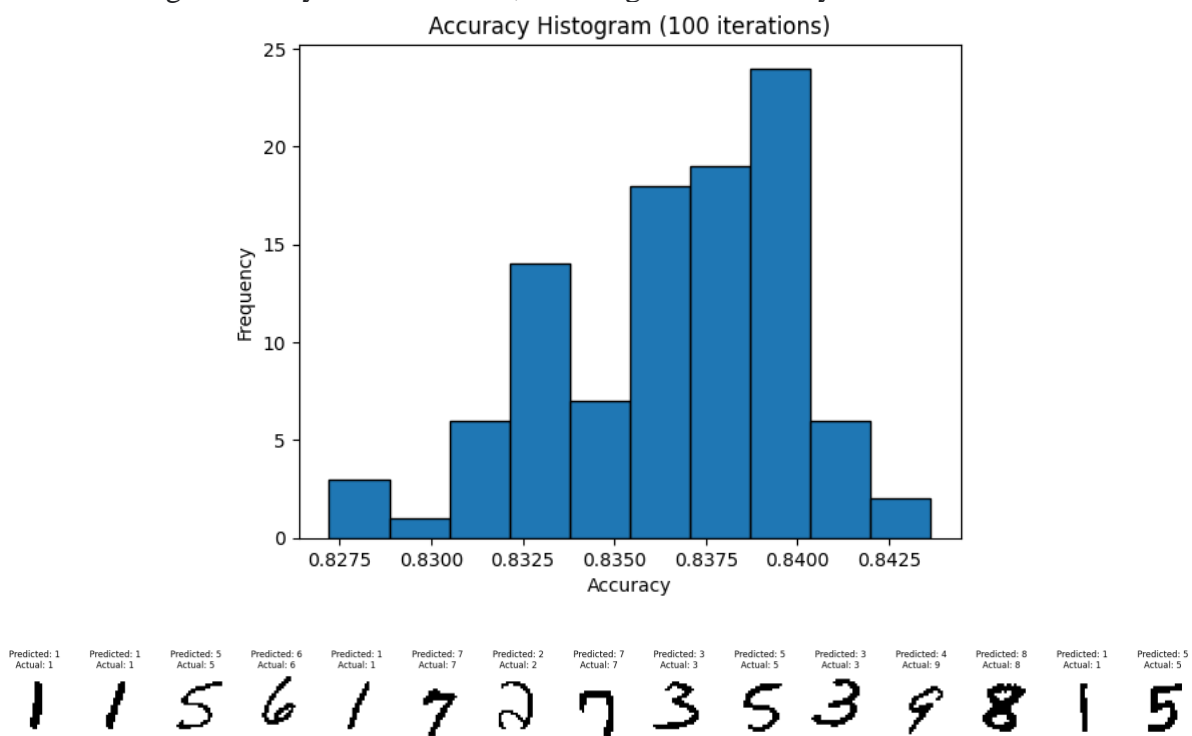
for i in range(num_iterations):
    X_train, X_test, y_train, y_test = train_test_split(X_binary, y, test_size=0.2, random_state=i)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    # Calculate accuracy and append to the list of accuracies
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

average_accuracy = np.mean(accuracies) * 100
print("Average Accuracy: {:.2f}%".format(average_accuracy))
```

[27] ✓ 2m 42.2s Python

... Average Accuracy: 83.66%

The test data was used to train the model, with random states varied to ensure diverse training scenarios. The accuracy of the model on each run was printed and visualized in a histogram and the average accuracy was calculated, resulting in an accuracy of 83.66%.



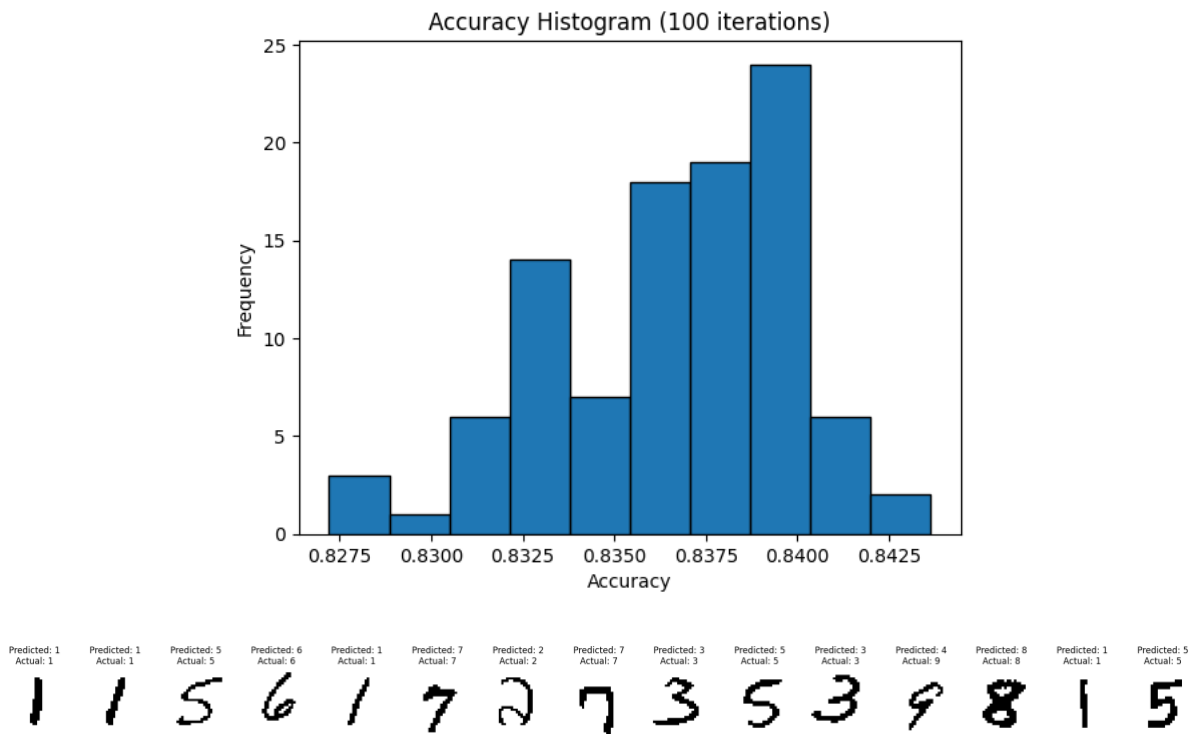
Fifteen examples of the process were visualized using the Matplotlib library.

Here are the detailed steps:

1. Begin by importing the necessary Python libraries.
2. Load the MNIST dataset and store it in a suitable data structure.
3. Convert the dataset from pandas DataFrames to NumPy arrays for further processing.
4. Apply binarization to the pixel values in the dataset.
5. Split the dataset into training and testing sets, using a test size of 20%
6. Create a custom class for the Bernoulli Naive Bayes model.
7. Initialize the model and train it using the training data.
8. Evaluate the model's performance by conducting 100 iterations.
9. Calculate accuracy for each iteration and compute the mean accuracy.
10. Visualize the distribution of accuracy using a histogram.
11. Visualize a few examples of this process using the Matplotlib library.

### 03. Experimental Results

The experimental results indicate that, on average, the implemented Bernoulli Naive Bayes model achieved an accuracy of 83.66% on the MNIST test dataset. This accuracy represents the proportion of correctly classified handwritten digits. To achieve a more accurate calculation of the model's accuracy, the model was trained iteratively. For each iteration, the resulting accuracy was visualized in a histogram, revealing that the majority of the accuracies clustered around the early 80% range. The accurately calculated accuracy also yielded a result of 83.66%. Additionally, the code includes a visualization of the model's predictions for a sample of 10 test images compared to their actual labels. This visual comparison helps understand how the model is performing on individual instances.



### 04. Discussion

This model takes Bernoulli Naive Bayes approach to simplify the MNIST problem, treating it as a binary classification task. It can be useful for the data that can be easily binarized and for simplification to problem solving offering straightforward solution to the problem.

One of the challenges in this project is that this model does not have high accuracy. It may be the inherent limitation that MNIST images are grayscale and not binary, and the pixel values' independence assumption may not hold. Consequently, there is a possibility that the information in the dataset may not be fully reflected. Additionally, it's worth considering other criteria beyond simply calculating the average accuracy for evaluating the model.