Yasser ISSAM, Christophe SAURY, Jung YSEOK

Introduction to Machine Learning Midterm Report

Problem Description:

For decades, digit identification has been a basic difficulty in the fields of computer vision and machine learning. The capacity to recognize and categorize handwritten numbers from a variety of sources, such as postal codes, bank checks, and forms, has applications in a wide range of sectors. The Modified National Institute of Standards and Technology (MNIST) dataset is one of the most famous and lasting datasets in this discipline.

Yann LeCun, Corinna Cortes, and Christopher J.C. Burges created the MNIST dataset, which has 70,000 handwritten digits, 60,000 in the training set and 10,000 in the test set. These digits are divided into 10 classes, each representing a number from 0 to 9. Each number is represented as a 28x28 pixel grayscale picture, with the intensity of each pixel quantized to a value between 0 and 255. Because of its simplicity and real-world applicability, the dataset is ideal for investigating and experimenting with categorization methods.

Target: 0
Target: 1
Target: 2
Target: 3
Target: 4
Target: 5
Target: 6
Target: 7
Target: 8
Target: 9
Target: 0
Target: 1
Target: 2
Target: 3
Target: 4
Target: 5
Target: 6
Target: 7
Target: 8
Target: 9

.

Theory and method chosen:

The Bernoulli Naive Bayes model, an extension of the more general Naive Bayes algorithm, is a probabilistic classification method widely employed in tasks involving binary features. It operates under the "naive" assumption that features are conditionally independent given the class, simplifying the computation of conditional probabilities. For the MNIST dataset, which can be transformed into binary pixel values, Bernoulli Naive Bayes is an attractive choice.

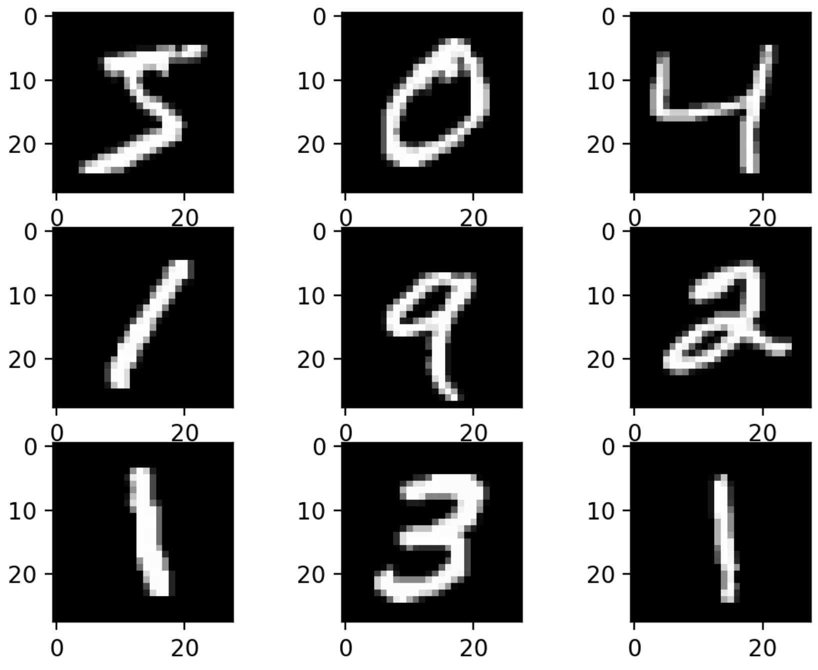
In the Bernoulli Naive Bayes model, each feature (in this case, each pixel) is treated as binary, where it is either "on" (1) or "off" (0). The model then calculates the likelihood of a feature occurring in each class (digit) and applies Bayes' theorem to compute posterior probabilities. This probabilistic approach makes it a valuable tool for classifying digits in the MNIST dataset, where we need to estimate the probability of pixel values given a specific digit class.

We decided to go with Bernouilli as a model because our data is represented into two different boolean values 0 or 1. Hence, the best probabilistic method that studies this case is the log-maximum likelihood of the Bernouilli Naive Bayes. It is more accurate to use the Bernouilli Naive Bayes than using a Multinomial Naives Bayes.

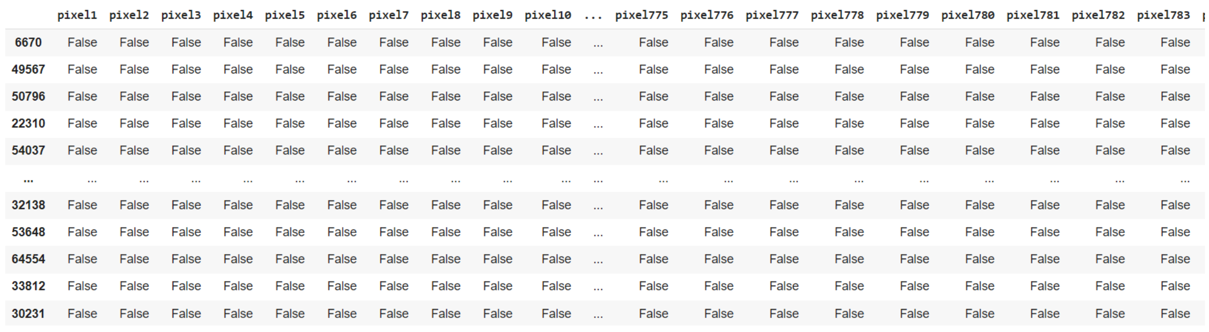
In this study, we set out to apply the Bernoulli Naive Bayes model on the MNIST dataset. Our goal is to get a better knowledge of its use, strengths, and limits in the context of digit recognition. We will go over our approach, trials, results, and discussions in great depth, providing insights into the performance and potential areas for improvement in this categorization assignment.

Implementation:

We fetch the MNIST dataset using the fetch_openml function.



The data is separated into different pixel values. We need to binarize the data. To do so, we convert the values that are classified from 0 to 256 to boolean values, so if you have a value from 128 to 256, you have the value true and from 0 to 128 you have the value False.

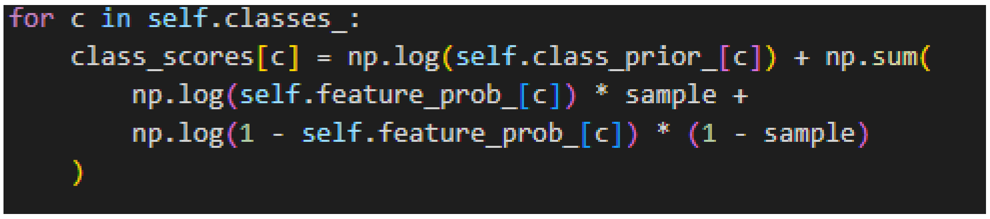


	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
6670	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
49567	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
50796	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
22310	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
54037	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32138	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
53648	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
64554	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
33812	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
30231	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False

Hence, we extract each target separately for binary classification, so if we have a picture of the number 5 per say, we will have two outputs, it is 5 or it isn't.

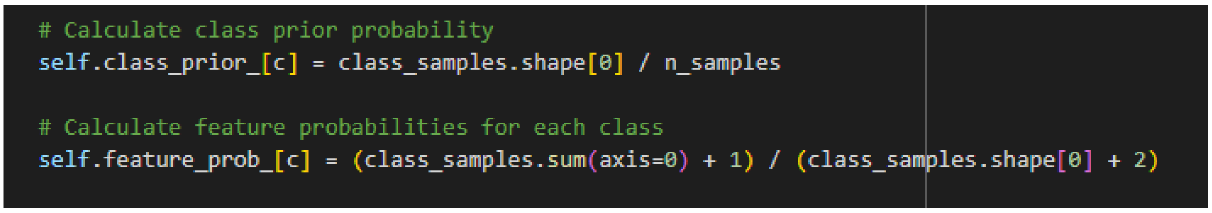
Our model computes the binary cross entropy loss to try to predict the class with the highest score. This is done with the following computation :

$$
\begin{aligned}
\hat{\mathbf{w}}_{ML} &= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1} \log p_{model}(y_i|\mathbf{x}_i, \mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log \left[\hat{y}_i^{y_i} \left(1 - \hat{y}_i\right)^{(1-y_i)}\right] \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{-y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)}_{\text{Binary Cross Entropy Loss } \mathcal{L}(\hat{y}_i, y_i)}
\end{aligned}
$$



```
for c in self.classes_:
    class_scores[c] = np.log(self.class_prior_[c]) + np.sum(
        np.log(self.feature_prob_[c]) * sample +
        np.log(1 - self.feature_prob_[c]) * (1 - sample)
    )
```

To apply this formula, we need to compute the prior class probability for each class and its feature probability.



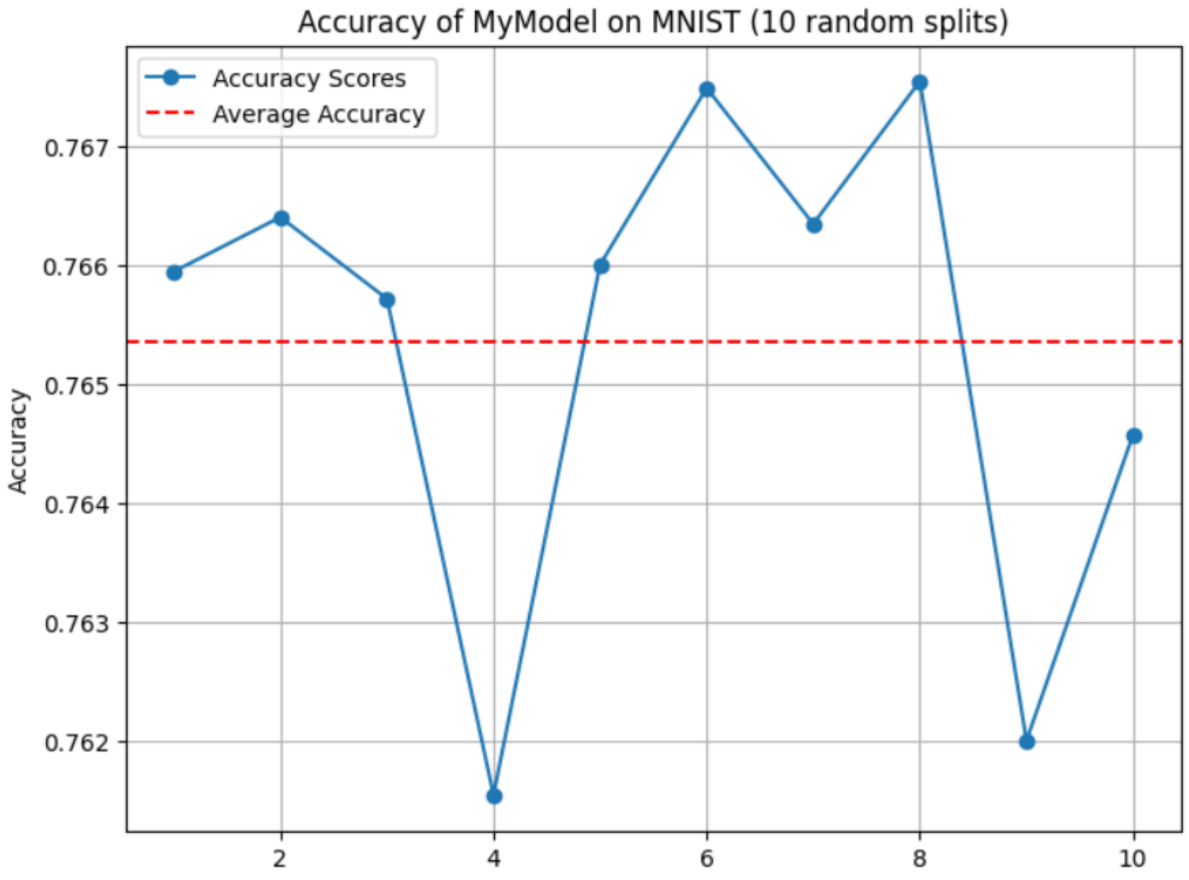
```
# Calculate class prior probability
self.class_prior_[c] = class_samples.shape[0] / n_samples

# Calculate feature probabilities for each class
self.feature_prob_[c] = (class_samples.sum(axis=0) + 1) / (class_samples.shape[0] + 2)
```

Experimental results:

Now that we have built a model that is supposed to recreate the Bernouilli Naives Bayes model, we need to test our model on the Mnist database.
We started off by splitting the Mnist data into a training set and a testing set. These training and testing sets are decided at random every time.

We test the accuracy of our model for 10 different training and testing sets and we get the average accuracy of our model over all 10 iterations. We get the following result :



We have an average accuracy of about 0.765.
This is a pretty good accuracy rate for our model but we wanted to find out if we could have gotten a better accuracy rate by implementing other models. We also want to compare it to the Bernouilli Naive Bayes model from the scilearn library.

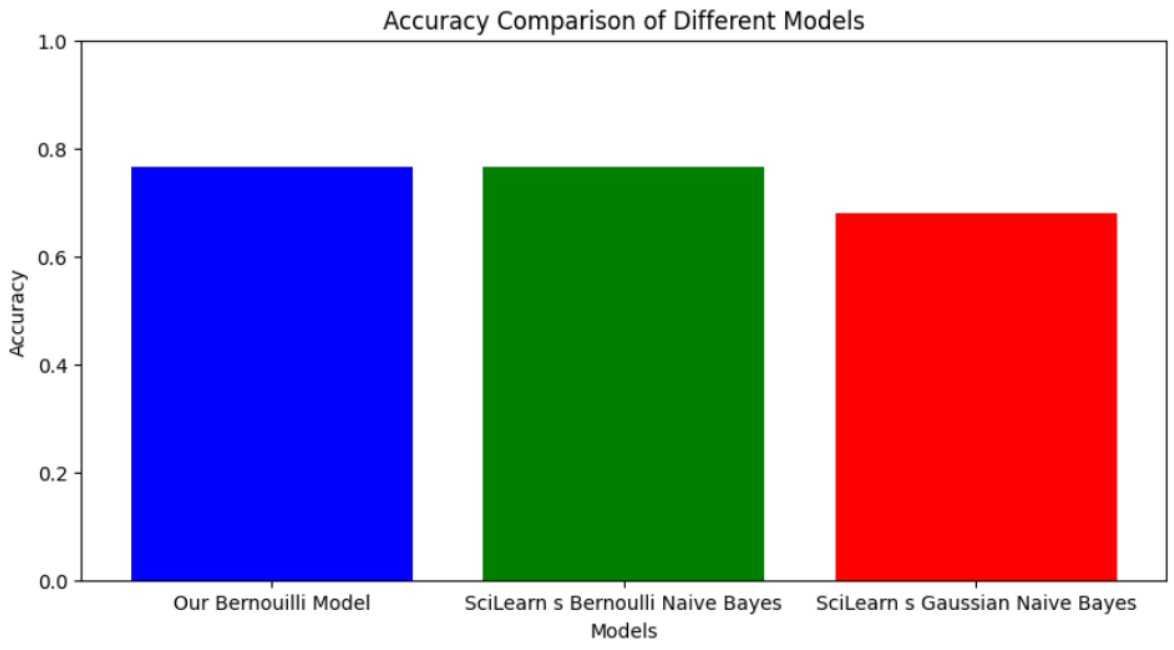
To do so we called on the other models and plotted their accuracies together with the accuracy of our model. We do so in the following code :

```
# Our bernouilli model
model = MyModel()
model.fit(Xtrain, ytrain)
accuracy1 = model.score(Xtest.values, ytest.values)
print(f"Testing accuracy: {accuracy1:.2%}")

# SciLearn's Bernoulli Naive Bayes model
model2 = BernoulliNB()
model2.fit(Xtrain, ytrain)
y_pred = model2.predict(Xtest)
accuracy2 = accuracy_score(ytest, y_pred)

# SciLearn's Gaussian Naive Bayes model
Xtrain_flatten = Xtrain.values.reshape(-1, 784)
Xtest_flatten = Xtest.values.reshape(-1, 784)
model3 = GaussianNB()
model3.fit(Xtrain_flatten, ytrain)
y_pred = model3.predict(Xtest_flatten)
accuracy3 = accuracy_score(ytest, y_pred)
```

This gives us the following result :



What this bar model shows us is that our custom model was implemented correctly as it is performing similarly to SciLearn's Bernouilli Naive Bayes. Our model also performs better than the general Gaussian Naive Bayes which comforts us in our decision to implement the Bernouilli Naive Bayes.

Discussion:

How to improve the model and why the result isn’t more accurate :

In the context of the MNIST dataset and many other image classification tasks, it's typically not accurate to assume that the pixel features are conditionally independent. The independence assumption is a key premise of the Naive Bayes model, including the Bernoulli Naive Bayes. However, in the case of images, especially those with spatial structures like MNIST, pixel values are not conditionally independent.

In the MNIST dataset, the neighboring pixels within an image are often correlated and provide contextual information about the digit being represented. For example, in a typical digit, the arrangement of pixels is such that a continuous stroke connects the digit's components. Assuming independence between neighboring pixels would ignore this spatial information and result in a significant loss of information.

The Naive Bayes model's conditional independence assumption simplifies the computation of probabilities but doesn't align with the natural dependencies in image data. This is one of the reasons why other machine learning models, such as convolutional neural networks (CNNs) are probably more suited to classifying images such as the ones in the Mnist Dataset.

Conclusion:

So, while the Bernouilli Naive Bayes model may be applied to the MNIST dataset for educational purposes, it's important to note that this independence assumption doesn't hold in practice, and more sophisticated models are better suited to handle the complexities of image data.