University of Mannheim

Business Informatics and Mathematics

Fall 2023

IE 675b Machine Learning

Dr. Rainer Gemulla

Assignment 1: Naive Bayes

October 15, 2023

Laila Albalkhi

Jonah Niklas Bjørgsvik Wiecek

lalbalkh | 1968154 | albalkhl@uwindsor.ca

jwiecek | 1966868 | jwiecek@students.uni-mannheim.com

Contents

1 Training

We calculate the priors with the following formula:

$$\frac{bincount(y) + (\alpha - 1)}{N + C * (\alpha - 1)} \tag{1}$$

where N is the size of our training set (60000), C is the size of our classes (10), and α is the parameter for the symmetric Dirichlet prior, using Laplace smoothing for all fitted distributions. We implement Laplace smoothing by adding $\alpha - 1$ to the numerator and multiplying the number of classes in the denominator by $\alpha - 1$. The numpy $bincount$ function counts the number of occurrences of each non-negative integer in an array. The $bincount$ function is much more efficient than standard array manipulation, significantly decreasing the algorithm's time complexity. In summary, this calculation counts the number of occurrences of each class label C in all labels y and normalizes the prior probabilities so that they sum up to 1. We store these prior probabilities in the array $priors$.

To calculate the class-conditional densities, we now use the following formula:

$$\sum_{c=0}^{C} \sum_{j=0}^{D} \frac{bincount(X_{y==c}, j) + (\alpha - 1)}{sum(y == c) + K * (\alpha - 1)} \tag{2}$$

where C is again the number of classes we have, D is the number of discrete features (784), K is the number of values the features can take (255), and X is the set of our training data. We use the $bincount$ function again here[1], this time counting the occurrences of all the different values of feature j with class c. We again normalize by the labels y with class c, taking into account Laplace smoothing once again. We store these class conditional densities in the array cls. Finally, our function nb_train returns a dictionary with two elements, with keys $logpriors$ and $logcls$ and their respective values.

1.1 Values of Alpha

When we run our function nb_train with an $\alpha = 1$, we effectively pass in a Dirichlet uniform prior. This raises an error in our program due to the zero-count problem, which arises when a feature value does not appear in the training set for a particular class. In our function, this raises a RunTime warning when dividing by 0, populating our class-conditional densities array with values of $-\infty$. We solve the above-mentioned problem by performing add-one smoothing, setting $\alpha = 2$. We now get the expected results for our class-conditional densities.

[1]We add a $minlength$ parameter equal to K to make sure that the output array is of compatible length, padding with 0s if necessary.

2 Prediction

For this section, we use the Naive Bayes assumption to calculate the predicted labels for each data set.

$$p(X, y = c) = \prod_{j=1}^{D} p(x_j|y = c) \cdot p(y = c) \tag{3}$$

Since we are working in the log space, we can apply the product rule of logarithms.

$$\log p(X, y = c) = \log \prod_{j=1}^{D} p(x_j|y = c) \cdot p(y = c) \tag{4}$$

$$= \sum_{j=1}^{D} \log p(x_j|y = c) + \log p(y = c) \tag{5}$$

Using the above, we can store the log joint distributions in the array $logjoint$ by adding the class conditional densities in a nested summation like the following:

$$\sum_{c=0}^{C} \sum_{j=0}^{D} logcls_{c,j,Xnew_{:,j}} + logpriors \tag{6}$$

where $logcls$ is the matrix from our previous task storing the class-conditional densities, $Xnew$ is a matrix with new data to predict the labels for, and $logpriors$ is the array from our previous task storing the class priors.

Now that we have the logarithm of the joint probability of each test sample and each class, we find the maximum value of the predicted class label for each class using the $argmax$ function and store it in the array $yhat$. We then create a new array, $logprob$ to store the log probability of the predicted class label for each table. Finally, we need to normalize in order to return the correct log probability of seeing the highest probable class for a training example. We use the $logsumexp$ function to prevent underflow issues while summing all the log probabilities in normalization[2] and return a dictionary with $yhat$ and $logprob$ values.

3 Experiments on MNIST digits data

3.1 Accuracy of the Model

The accuracy of our model is 0.8363, or approximately 84%.

[2]Note that we have to transpose the array $logjoint$ because the $logsumexp(x)$ function provided is defined to compute the $logsumexp$ over each column when x is a two-dimensional array.

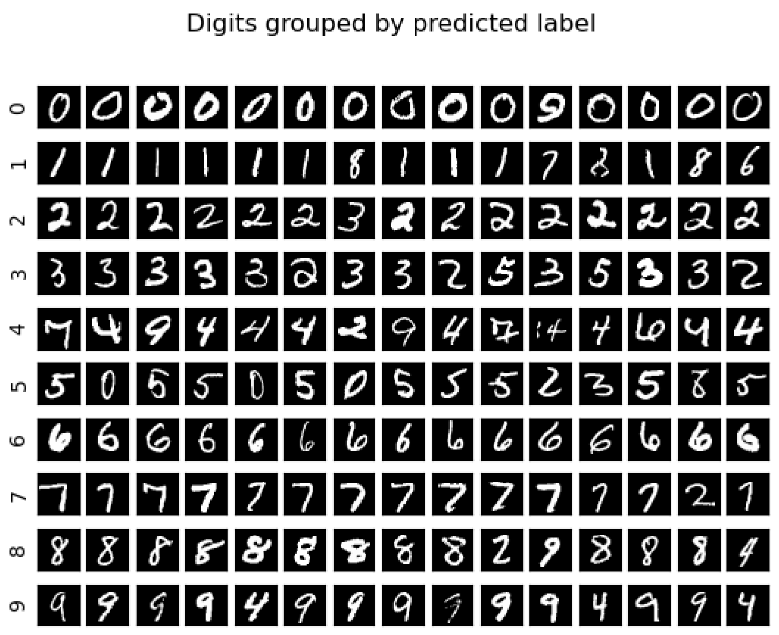

Figure 3.1: 150 digits grouped by their respective predicted labels, as determined by the Naive Bayes model using $\alpha = 2$.

3.1.1 Errors and Confusion Matrix

Figure 3.1 shows us 150 digits and their respective predictive labels as determined by our Naive Bayes model. At first glance, we notice some misclassification errors. We see a 9 misclassified as a 0, for instance, and a few 4s misclassified as 9s. For this specific random subset of the data, we don't see any errors for the digit 6, and minimal errors for the digit 7. When we plot only the misclassified digits in Figure 3.2, the errors start to become clearer. Here we can see that 5s are often misclassified as 3s, and 0s are often misclassified as 5s. Moreover, we see that the machine often confuses 3s and 8s, labelling them inversely. We gain more insight into the error proportions from Figure 3.3, where we can see that the majority of errors for the random sample of data we pulled are often seen in the classes for digits 3 and 9. The digit 7 has a low misclassification proportion, with only 4 errors shown for this specific sample. The digit 5 has a low misclassification rate as well, with only 6 errors shown.

We get a comprehensive overview of all data in our training data by using the classification report as shown in Figure 3.4. The precision in the classification class refers to the fraction of true positives among the samples predicted as positives, including false positives. Recall refers to the fraction of true positives among the samples that are actually positive. Precision measures the accuracy, whereas recall measures the completeness. The F1-score is a weighted average of both precision and recall and is maximized when the two metrics are equal.

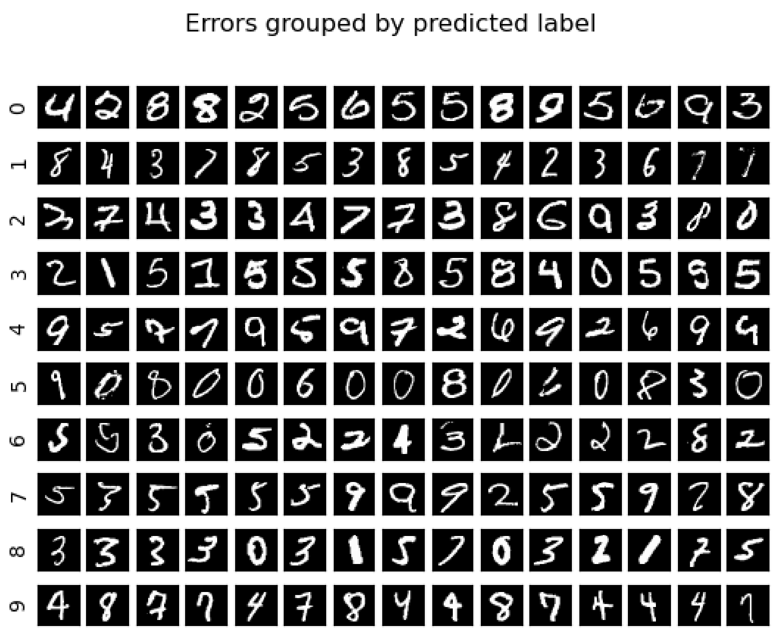

Figure 3.2: 150 misclassified digits grouped by their respective wrong labels, as determined by the Naive Bayes model using $\alpha = 2$.

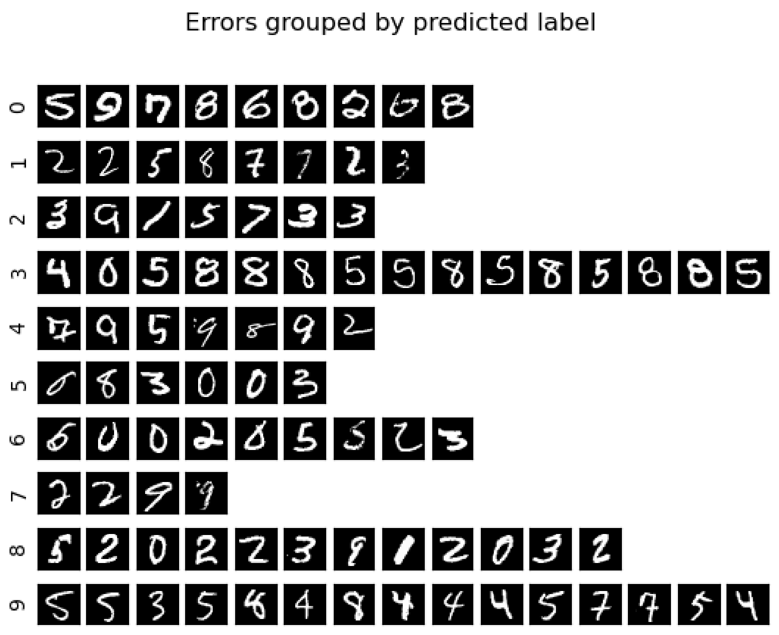

Figure 3.3: Random sample of error proportions grouped by their respective label as determined by the Naive Bayes model using $\alpha = 2$.

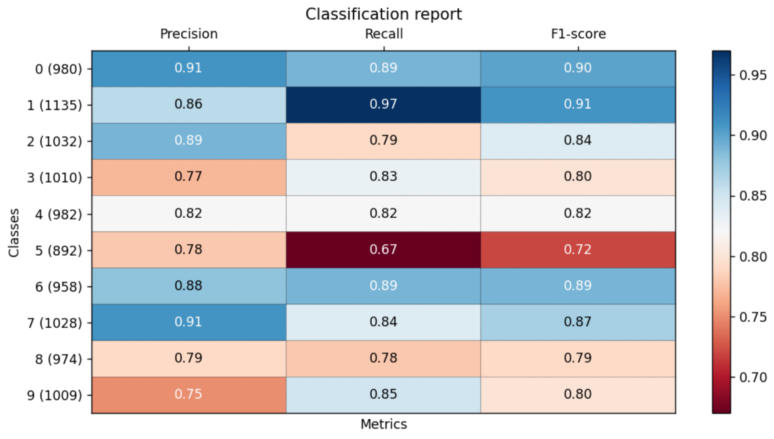

Figure 3.4: The classification report for the predicted labels plotted onto a heatmap.

From the classification report, we can see that class 5 has the lowest recall score, whereas class 1 has the highest. Classes 0, 7, and 1 have the highest precision rates, whereas 9 has the lowest. We can look into this in further detail by using the confusion matrix. The diagonal value in our confusion matrix indicates the number of true positives; accurate predictions that match the true label. The sum of values of the corresponding rows would yield the number of false negatives, and the sum of values of the corresponding columns would yield the number of false positives. In general, we see that these metrics also correlate with the earlier figures for predicted class labels.

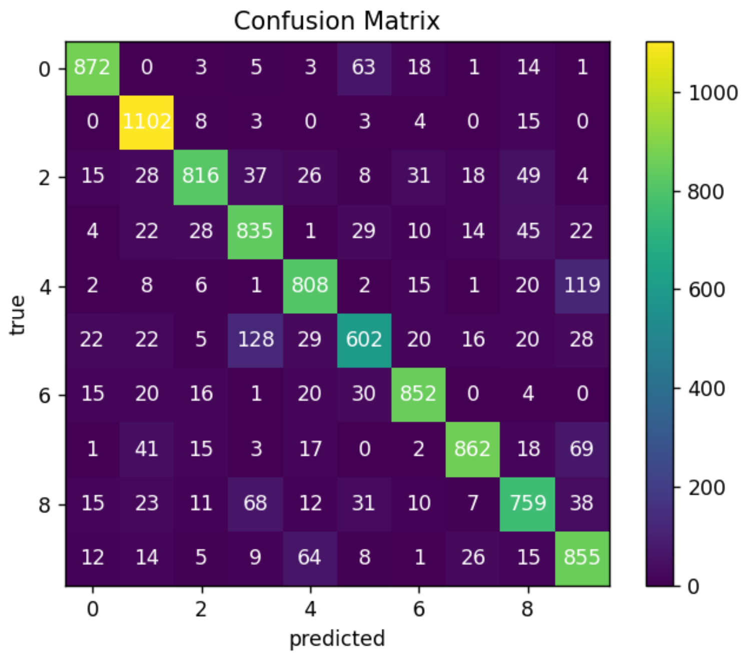

Figure 3.5: The confusion matrix of the predicted and true labels of the dataset.

4 Model Selection

For model selection or hyperparameter optimization, we use Kfold cross-validation with K equal to five. We do this iteratively to try to find a good estimator for the optimal α. We go through a loop with a start and end $alpha$ and define a variable $stepsize$. We then loop until we find a max accuracy that has a difference to its neighbouring $alphas$ (in the array) that is less than a specific stopping criterion. After each loop we half the step size and shrink the range.

In more detail, we set the stopping criterion to 0.05% and set start and end values of α to 0.5 and 2.5 respectively. We choose a $stepsize$ of 0.5, so our first iteration of cross-validation is with the following array for α: $alphas = [0.5, 1.0, 1.5, 2.0, 2, 5]$. After the first iteration, we find the highest average accuracy to be with $\alpha = 1.5$. We then run from the points before and after the optimal α with half the $stepsize$. The new array becomes $alphas = [1.0, 1.25, 1.5, 1.75, 2.0]$. This continues until the stopping criterion is met.

As you can see from the two first arrays for α, there are three duplicate values. In order to reduce the computing needed, we can utilize memoization. To achieve this, we have added a dictionary that caches the accuracy calculated with kfold cross-validation for each value of α already seen. Please see the code for further implementation details. Our code stops after 4 iterations and returns the best value for α found, which in our implementation is 1.125. We then trained with $\alpha = 1.125$ on the whole training set and got an accuracy of 0.8463 on the test set.

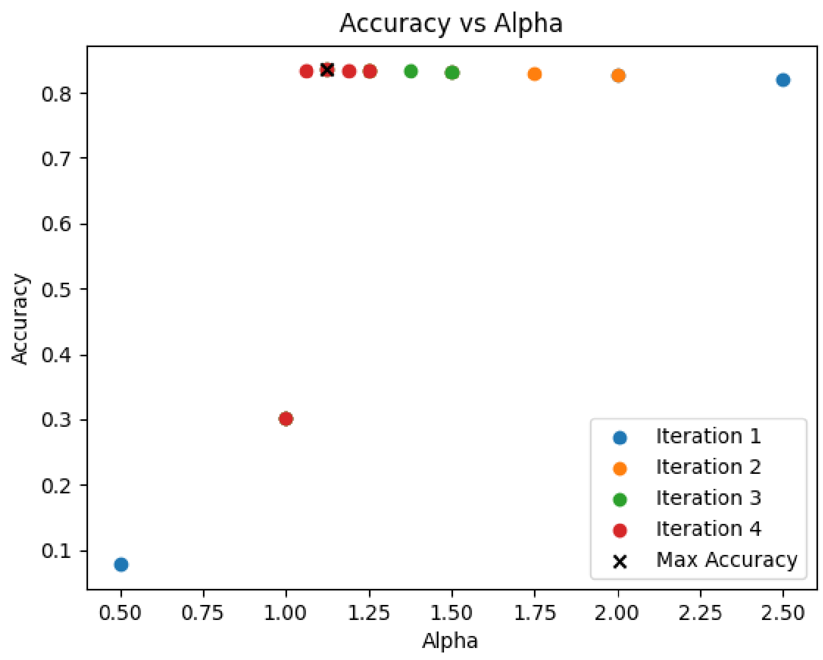

Figure 4.1: Alphas used in the iterative cross validation and their accuracies

5 Generating Data

5.1 Generating Digits

We generate the digits for a class c using the class-conditional densities previously computed and stored in $logcls$. We exponentiate the logarithmic value of the class-conditional densities to obtain the estimates, resulting in an array $theta$ of D features and K possible feature values. We then iterate over each feature j, generating a random feature value given by the j-th row of $theta$. The resulting array $Xgen$ contains the synthetic data for a given class c. The expected values clearly show us the digits 0-9, see figure 5.1.

5.2 Hyperparameters

We then tried generating numbers with different values for α. First, we observed that we got an error with $\alpha < 1$, because class, feature, and value combinations with zero count result in negative probabilities. For this reason, we experimented with values of $\alpha >= 1$.

We quickly noticed that as α increases, the generated images become "noisier"; the number of white dots outside of the "digit shape" seem to increase, see figure 5.3. Since we use a uniform Dirichlet prior, α does not affect the most likely value of each feature per class. Since we add the same value to all the probabilities, the maximum value will stay as is. For the expected value, the image should get smudged out because the probability distribution over the values for a specific feature and class gets smoothed out with a higher α. In other words; the expected digit will get "whiter". With our value testing of $\alpha = 5$, we didn't really see a difference in comparison to the optimal α, but when we increased the value we saw a clear diffusion or whitening of the visualized expected values.

6 Missing Data

6.1 First Formula

Using the Bayes rule we get:

$$p(y|x_{1:D}) = \frac{p(x_{1:D}|y) \cdot p(y)}{p(x_{1:D})} \tag{7}$$

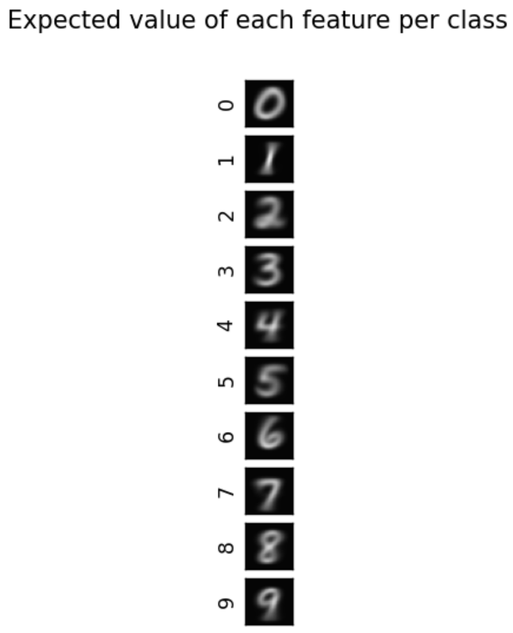

Figure 5.1: Expected values of the digits grouped by the class labels.

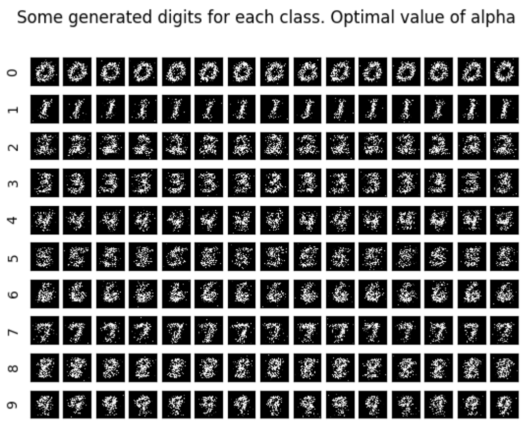
Some generated digits for each class. Optimal value of alpha
0
1
2
3
4
5
6
7
8
9

Figure 5.2: Generated digits with $\alpha = 1.125$

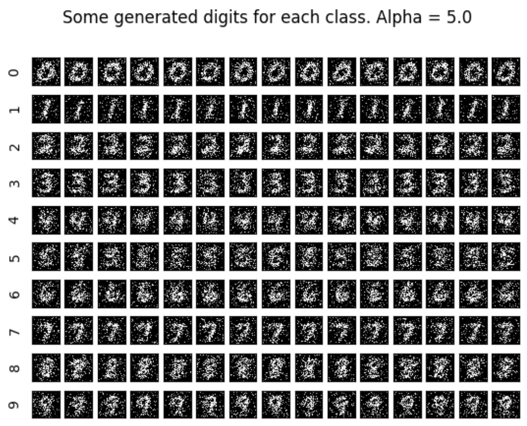

Figure 5.3: Generated digits with $\alpha = 5.0$

Then we can use the Naive Bayes assumption:

$$= \frac{\prod_{j=1}^{D} p(x_j|y) \cdot p(y)}{p(x_{1:D})} \tag{8}$$

$$= \frac{\prod_{j=1}^{D} p(x_j|y) \cdot p(y)}{\sum_{y=0}^{C} p(x_{1:D}, y)} \tag{9}$$

$$= \frac{\prod_{j=1}^{D} p(x_j|y) \cdot p(y)}{\sum_{y=0}^{C} \prod_{j=1}^{D} p(x_j|y) \cdot p(y)} \tag{10}$$

$$= \frac{p(y) \cdot \prod_{j=1}^{D} p(x_j|y)}{\sum_{y=0}^{C} p(y) \prod_{j=1}^{D} p(x_j|y)} \tag{11}$$

Note that $p(y)$ is not a part of the product Π.

$$p(y|x_{1:D}) \propto \prod_{j=1}^{D} p(x_j|y) \cdot p(y) \tag{12}$$

In classification tasks, this formula is very useful when we want to maximize $p(y|x_{1:D})$. Since we know the class conditional density and the prior we don't need to calculate $p(x_{1:D})$. In particular, this is useful when we want to estimate the probability of a particular class given a set of observed features.

6.2 Second Formula

This is the exact same as task a) just that we don't have all the features j from 1 to D, but we only have the features from 1 to D'. Using the Bayes rule:

$$p(y|x_{1:D'}) = \frac{p(x_{1:D'}|y) \cdot p(y)}{p(x_{1:D'})} \tag{13}$$

And now using the Naive Bayes assumption:

$$= \frac{\prod_{j=1}^{D'} p(x_j|y) \cdot p(y)}{p(x_{1:D'})} \tag{14}$$

$$= \frac{\prod_{j=1}^{D'} p(x_j|y) \cdot p(y)}{\sum_{y=0}^{C} p(x_{1:D'}, y)} \tag{15}$$

$$= \frac{\prod_{j=1}^{D'} p(x_j|y) \cdot p(y)}{\sum_{y=0}^{C} \prod_{j=1}^{D'} p(x_j|y) \cdot p(y)} \tag{16}$$

$$p(y|x_{1:D'}) \propto \prod_{j=1}^{D'} p(x_j|y) \cdot p(y) \tag{17}$$

This is useful when we only want to use a subset of the features we believe to be relevant for classification, or if we want to reduce the complexity or computational power required to predict all classes by ignoring less relevant features and data points. This will simplify the model, but could potentially lead to a loss of accuracy depending on the features we choose to disregard.

6.3 Third Formula

$$p(x_{1:D'}|x_{D'+1:D}) = \frac{p(x_{1:D'}, x_{D'+1:D})}{p(x_{1:D'})} \tag{18}$$

$$= \frac{p(x_{1:D})}{p(x_{1:D'})} \tag{19}$$

Using the sum rule:

$$= \frac{\sum_{y=0}^{C} p(x_{1:D}, y)}{\sum_{y=0}^{C} p(x_{1:D'}, y)} \tag{20}$$

Using the definition of conditional probability and the Naive Bayes assumption:

$$= \frac{\sum_{y=0}^{C} \prod_{j=1}^{D} p(x_j|y) \cdot p(y)}{\sum_{y=0}^{C} \prod_{j=1}^{D'} p(x_j|y) \cdot p(y)} \tag{21}$$

This form is useful when we only have the priors and class-conditional densities of the dataset, and we want to calculate the probability of missing data. We can use this to predict the values of missing data points given only the conditional distribution and priors.

7 Conclusion

In this assignment, we explored the effectiveness of the Naive Bayes model given a dataset of handwritten digits. Our findings have proved that despite the simplicity of the Naive Bayes model and the conditional independence assumption it makes, it surprisingly yields accurate results when classifying handwritten numbers into digits, of approximately 85%. By leveraging the prior distribution, the class-conditional densities, and exploring optimal values of the α hyperparameter, we were even able to generate digits using the model for each class label. We can generate the expected values for each class with a fair accuracy. While Naive Bayes is nowhere near a perfect model, it is an efficient and robust method to process and classify data in cases where feature independence is a fair assumption.