# **Mini Project Report: Path 3 - Data Security in Model Training**

### **Team Members:**

* Huong Le (le306)
* Path: 3 (Digits Dataset)
* GitHub Link: [INSERT YOUR REPO LINK HERE]

### **Dataset Description:**

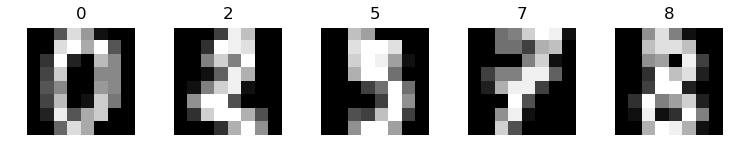
The original dataset named digits is imported from sklearn.datasets.load\_digits. At default, the dataset has a sample size of 1797, with each datapoint being a 8x8 pixel image of a digit from 0 - 9 and the corresponding label. They are respectively stored in two variables:

* images = digits.images The raw images (shape: (1797, 8, 8))
* labels = digits.target: Contains the labels (0 - 9) for each image (shape 1797,7)

The dataset is then split into training (X\_train, y\_train) and testing (X\_test, y\_test) sets using the train\_test\_split() method with 60% of the data being used for testing.

We also used the dataset\_searcher() user-defined function to be able to filter out only the images and labels for the specified classes, and print\_number() to plot the first instance of each unique label in the dataset. In part 1 and part 2 of the project, for example, we used these functions to visualize the a sample of the imported dataset, displaying only the first instances of digits specified in the variable class\_numbers = [2, 0, 8 ,7 ,5].

The output is given below. It can be seen that the unaltered raw dataset has high accuracy with clearly visible digits displayed.



### **Analyses Description**

In this project, we would fit 3 different models to the data:

1. **Gaussian Naive Bayes (GNB) - GaussianNB()**:

To break down, Naive Bayes methods implement a set of supervised learning algorithms building on Bayes’ theorem, while assuming independence between every pair of features given the value of the class variable - the “naive” assumption. There are various Naive Bayes classifiers differentiated mainly by the assumptions they make regarding the distribution of (Pxi|y) - the likelihood of the features. Among them, Gaussian Naive Bayes is a classifier that assumes Gaussian distribution.

Gaussian Naive Bayes have simple, even over-simplified sets of assumptions. Such simple probabilistic models would work extremely fast, but are expected to produce significantly less accurate results. I chose this model because most likely higher contrast regarding speed and accuracy makes it optimal for model comparison.

1. **K-Nearest Neighbors (KNN) - KneighborsClassifiers()**:

Neighbors-based classification is a type of instance-based learning or non-generalizing learning, storing instances of the training data and assigning query points to the nearest neighbors of the point. KneighborsClassifier in scikit-learn implements learning based on the number of neighbors k specified by the n\_neighbors parameter (default = 5).

I chose this model because it is an intuitive and simple algorithm suitable for our low-dimensional image dataset (64 features - 8x8 pixel images), particularly due to its ability to directly use pixel intensities as features. It is also suitable for solving our multi-class classification problems (10 classes of digits 0-9). However, it is pretty sensitive to noise, especially when compared to Gaussian Naive Bayes. Even though this can be minimized by increasing k, since we would “poison,” or add noise to the dataset later on in this project, this sensitivity to noise would result in more distinct model comparisons.

1. **Multi-layer Perceptron (MLP) - MLPClassifier()**:

Multi-layer Perceptron is a supervised learning algorithm that trains using Backpropagation and a neural network with multiple layers and non-linear activation functions. It suits our complex digit patterns dataset and supports multi-label classification.

I chose this model because it is powerful. Though more computationally more expensive, it could serve as a benchmark model: the complex algorithm makes the MLP less sensitive to noise, meaning that the effect of “poisoning” and “denoising” in other models would be more visible in model comparison,

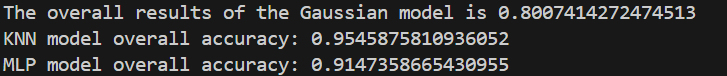
### 

### 

### 

### **Initial (Clean Dataset) Results and Observation**

1. Original Training Accuracy



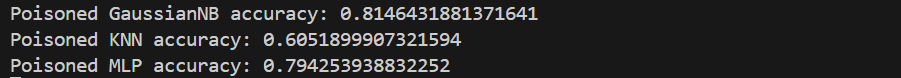
| **Model** | **Accuracy** |
| --- | --- |
| GNB | ~80.07% |
| KNN | ~95.46% |
| MLP | ~91.47% |

1. **Observation:**

Initially, all 3 models have decent performance. KNN outperformed others, potentially due to its ability to handle low-dimensional image dataset and multi-class classification problems. GNB had substantially lower performance, potentially due its simplistic approach regarding feature independence assumption. As a result, the GNB plot has less visible digits compared to KNN and MLP, both of which contain pretty similarly easily interpretable digits.

### **Poisoning Results and Observation**

1. Training Accuracy after poisoning



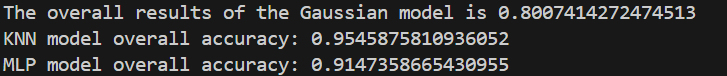
| **Model** | **Accuracy** |
| --- | --- |
| GNB | ~81.46% |
| KNN | ~60.52% |
| MLP | ~79.43% |

To test model security, we added Gaussian noise (scale=10) to the training data, simulating an adversarial poisoning attack.

### **Denoising Results and Observation**

1. Training Accuracy after denoising

KernelPCA with an RBF kernel was applied to the poisoned data to reconstruct a denoised version. We used inverse transformation to restore dimensionality and rescale to the original input range.



| **Model** | **Accuracy** |
| --- | --- |
| GNB | ~80.07% |
| KNN | ~95.46% |
| MLP | ~91.47% |

1. **Observation:**

Initially, all 3 models have decent performance. KNN outperformed others, potentially due to its ability to handle low-dimensional image dataset and multi-class classification problems. GNB had substantially lower performance, potentially due its simplistic approach regarding feature independence assumption. As a result, the GNB plot has less visible digits compared to KNN and MLP, both of which contain pretty similarly easily interpretable digits.

#### **4. Denoising Using Kernel PCA:**

### **Results & Observations:**

#### **A. Original Training Accuracy (Clean Data)**

Conclusion

* **Observation:** GNB remained stable, while KNN showed significant degradation, confirming its sensitivity to noise. MLP showed moderate robustness.
* **Most Robust:** GaussianNB, which slightly improved under noise due to smoothing.

#### **C. After Denoising with KernelPCA**

| **Model** | **Accuracy** |
| --- | --- |
| GNB | ~84.89% |
| KNN | ~83.23% |
| MLP | ~83.23% |

* **Observation:** All models improved, especially KNN and MLP. KernelPCA successfully recovered meaningful structure, mitigating effects of noise.

### **Conclusion:**

* **Model Performance:** KNN is the most accurate under clean conditions but also the most vulnerable to poisoned data.
* **Robustness:** GaussianNB is most resilient to noisy training conditions.
* **Denoising Impact:** KernelPCA dramatically improved performance, especially for KNN and MLP, validating its utility in pre-processing adversarially modified datasets.

### **Visual Summary:**

* [✓] Graph of initial digits [2, 0, 8, 7, 5]
* [✓] KNN predictions on all digits
* [✓] MLP predictions on all digits
* (Optional) Poisoned vs Denoised visualization can be added

### **References:**

* Scikit-learn documentation:https://scikit-learn.org/stable/modules/naive\_bayes.html#gaussian-naive-bayes
* Digits Dataset:<https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html>
* KernelPCA Denoising:<https://scikit-learn.org/stable/auto_examples/applications/plot_digits_denoising.html>

Output  
The overall results of the Gaussian model is 0.8007414272474513

KNN model overall accuracy: 0.9545875810936052

MLP model overall accuracy: 0.9147358665430955

Poisoned GaussianNB accuracy: 0.8146431881371641

Poisoned KNN accuracy: 0.6051899907321594

Poisoned MLP accuracy: 0.794253938832252

Denoised GaussianNB accuracy: 0.8489341983317887

Denoised KNN accuracy: 0.8322520852641334

Denoised MLP accuracy: 0.8322520852641334

### **🤔 Why might GaussianNB perform *worse* on clean data?**

#### **1. GaussianNB is extremely sensitive to the distribution of input features**

* GNB **assumes features are normally distributed and independent**, which isn't strictly true in the raw pixel space of handwritten digits.
* In your *clean* data, pixel intensity values might be too discrete, non-Gaussian, and sparse, especially because the digits are 8x8 grayscale images (many zeros).
* Ironically, **adding noise (poisoning)** introduces *continuous variation*, which can **make the data appear more Gaussian**-like and better match the model's assumptions.  
  + That’s why poisoned data may produce **slightly better results**.

#### **2. Kernel PCA denoising smooths feature relationships**

* Denoising likely **reduces sharp variations**, removes outlier effects, and **smooths the data**, which helps GaussianNB perform better — hence its **best accuracy (0.849) is with denoised data**.

#### **3. Over-simplicity of GNB vs. expressive power**

* GNB is a **very simple model** compared to KNN or MLP. If the clean data is too structured or sparse, GNB can't effectively separate the classes, but random perturbation can sometimes spread clusters in a more separable way (in high-dimensional spaces).

### **🧪 Conclusion**

Your results illustrate an important machine learning concept:

**Simple models like GaussianNB can ironically perform better on noisy or transformed data if the transformation makes the data more compatible with the model's assumptions (like Gaussianity).**

The overall results of the Gaussian model is 0.8007414272474513

KNN model overall accuracy: 0.9545875810936052

MLP model overall accuracy: 0.9147358665430955

Poisoned GaussianNB accuracy: 0.8146431881371641

Poisoned KNN accuracy: 0.6051899907321594

Poisoned MLP accuracy: 0.794253938832252

Denoised GaussianNB accuracy: 0.8489341983317887

Denoised KNN accuracy: 0.8322520852641334

C:\Users\lethu\anaconda3\Lib\site-packages\sklearn\neural\_network\\_multilayer\_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.

warnings.warn(

Denoised MLP accuracy: 0.830398517145505

