Bayes Classifiers and Their Application in Ground Truth Annotation

A Detailed Exploration of Classification Techniques and Real-World Use Case

Presented by: Alham Hotaki

Date: 03.September.2024

Introduction to MNIST Dataset

What is MNIST?

- The MNIST dataset is a benchmark dataset in the machine learning community, consisting of 70,000 images of handwritten digits (0-9).
- Each image is 28x28 pixels, represented as grayscale values.

Why MNIST?

- MNIST is widely used because it is simple yet challenging, providing a standard baseline for evaluating algorithms.
- It is particularly useful for testing classification algorithms and deep learning models.

What is Naive Bayes Classifier?

Imagine you're trying to guess what kind of fruit someone is thinking about based on clues like color, size, and taste. If you know that apples are usually red and sweet, and bananas are yellow and sweet, you can make a pretty good guess. This is basically what the Naive Bayes Classifier does.

How does it work?

- It looks at the features (like color, size, taste) and guesses which category (like apple, banana, orange) the item belongs to.
- It's called "naive" because it assumes that each feature (like color and size) is independent of the others, even though that's not always true. For example, the classifier might assume that just because something is red, it doesn't necessarily have to be round, which might not be accurate for fruits.

What is Gaussian Naive Bayes Classifier?

Now, let’s make things a little more precise. Imagine you know that not only are apples usually red, but they’re also usually a certain size and shape. The Gaussian Naive Bayes model doesn’t just look at general features like "red" or "sweet," but it also considers that these features follow a certain pattern or distribution.

Gaussian means:

- The model assumes that the features (like the size of the fruit or the brightness of the X-ray) follow a normal distribution. This is a fancy way of saying that most values are around an average, with fewer and fewer values as you move away from this average (like how most people are of average height, with fewer being very short or very tall).

What is Kernel Density Estimation (KDE)?

Imagine you're trying to guess what kind of fruit someone is thinking about, but this time the clues don't follow a simple pattern. Maybe the fruits are exotic, and their colors and sizes are all over the place. The KDE Classifier is like a super-flexible tool that doesn't assume anything about the patterns but tries to "feel out" the data in a smooth way.

How does it work?

- Instead of assuming a specific shape for the data (like the bell curve in Gaussian), KDE looks at the data points you have and estimates where most of them are, like a smooth curve that fits around the data points.
- It's like looking at a bunch of dots on a map and drawing a smooth, wavy line that goes around where most of the dots are concentrated.

Analogy to Summarise:

Naive Bayes: Imagine you're guessing someone's favorite fruit based on simple clues like color and taste, assuming these clues don't influence each other.

Gaussian Naive Bayes: You make a more precise guess because you know the clues follow a certain pattern (like most apples are a similar size and color).

KDE Classifier: You throw away all assumptions and just try to fit a smooth curve around the data to make the best possible guess, no matter how irregular the data might be.

In all cases, these classifiers are helping to make predictions based on what they've learned from past examples, whether it's identifying fruits, diagnosing diseases from medical images, or any other classification task.

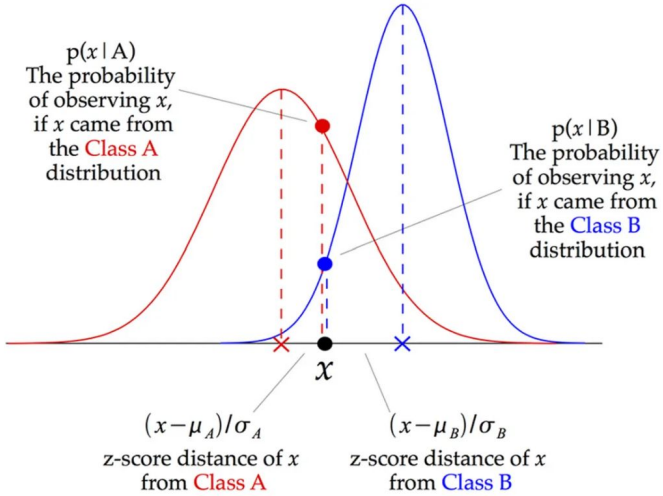
Gaussian Naive Bayes Classifier

Implementation:

- Mean and variance are calculated for each feature across all classes.
- Posterior probabilities are computed for classification.

Performance:

- Accuracy: 63.23%
- Pros: Simplicity, speed
- Cons: Assumes feature independence, lower accuracy



Multivariate Gaussian Bayes Classifier

Implementation:

- Calculates mean vector and covariance matrix for each class.
- PCA used to reduce dimensionality to 50 components.

Performance:

- Accuracy: 96.39%
- Pros: Accounts for feature correlation, better accuracy
- Cons: Computationally intensive, sensitive to outliers

KDE Bayes Classifier

Implementation:

- KDE with Gaussian kernel and bandwidth parameter.
- PCA used to reduce dimensionality.

Performance:

- Accuracy: 97.33%
- Pros: Flexible, high accuracy
- Cons: Requires careful selection of bandwidth, computationally demanding

Comparative Analysis

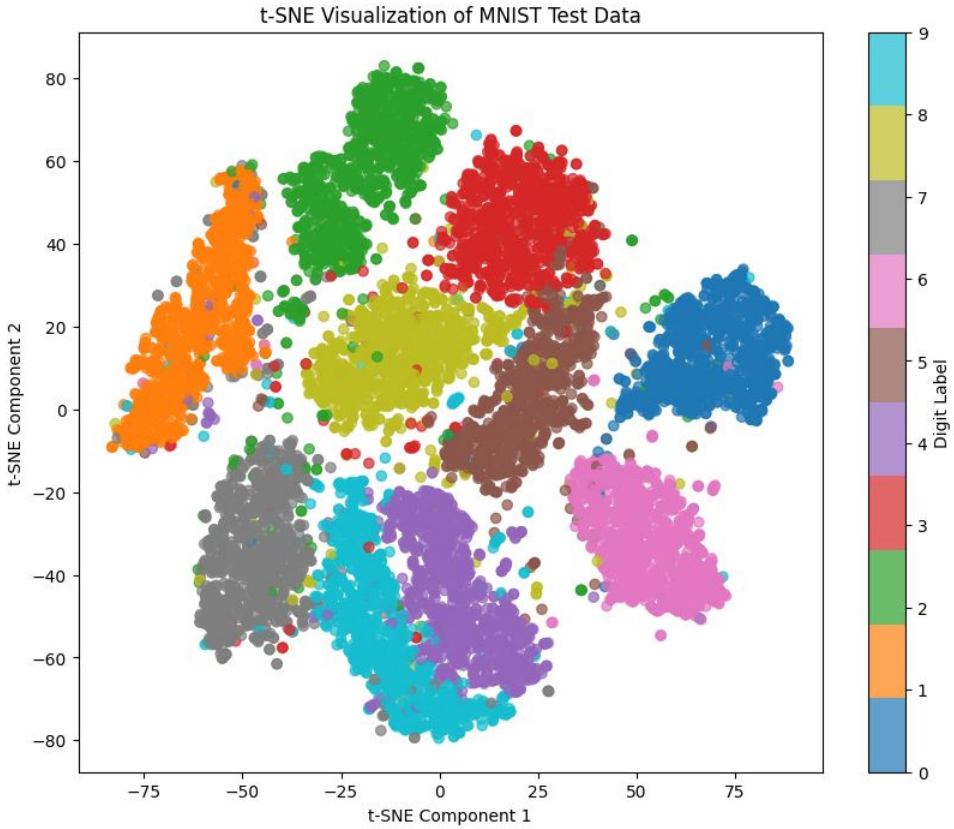
Accuracy Comparison:

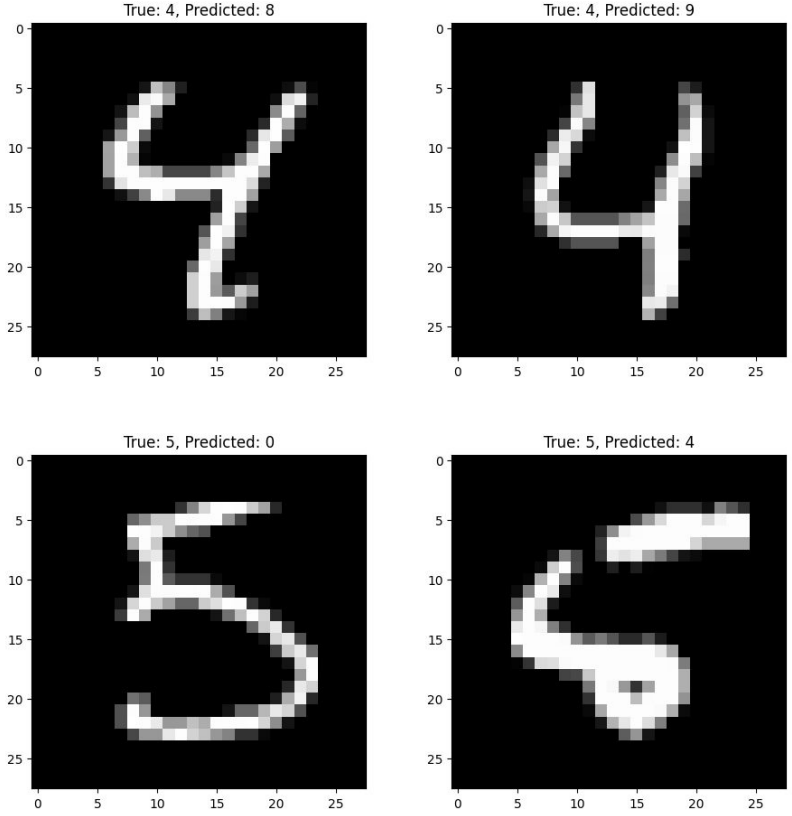
- Gaussian Naive Bayes: 63.23%
- Multivariate Gaussian Bayes: 96.52%
- KDE Bayes: 97.36%

Conclusions:

- KDE-based Naive Bayes offers the highest accuracy.
- Dimensionality reduction with PCA improves computational efficiency.
- Choice of model depends on the trade-off between accuracy and computational resources.

Let’s Visualise the MNIST Dataset



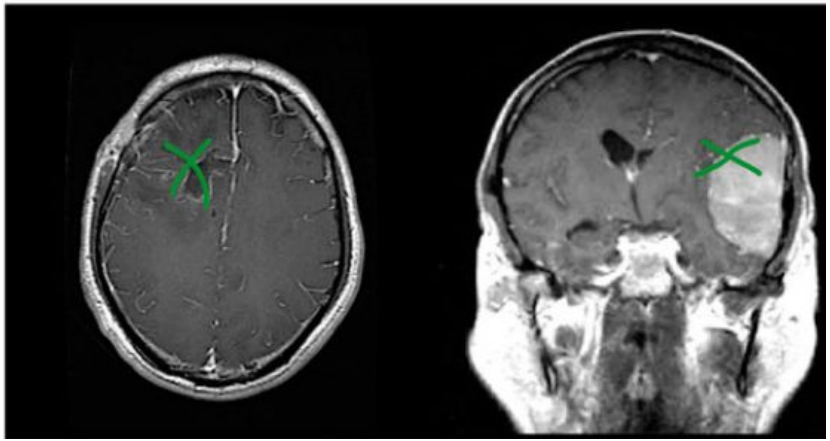


Use Case

Ground Truth Annotation

What is Ground Truth Annotation?

Lets imagine that we are developing a computer program that can look at medical images (like X-rays or MRIs) and help doctors diagnose diseases, such as detecting tumors in the brain. This is where ground truth annotation becomes crucial.



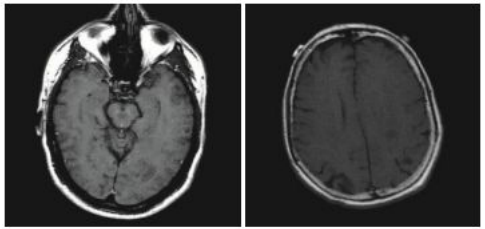
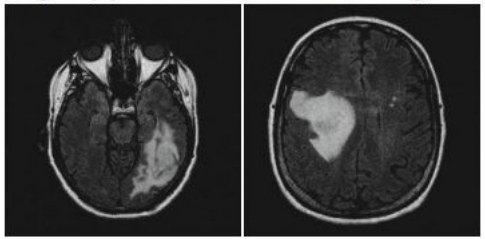


Fig 2. (a) Normal brain MRI Image



(b) Abnormal brain MRI Image

Reference: https://www.irjet.net/archives/V7/i6/IRJET-V7I61270.pdf

How to build a Model inspired by MNIST

Step 1: Collecting Data

Doctors collect thousands of brain scans from patients, including images of healthy brains and brains with tumors.

Step 2: Ground Truth Annotation

To teach the computer what a tumor looks like, doctors and medical experts have to go through these brain scans and label the tumors. This means they might draw circles around the tumors and label them as "tumor" or "no tumor." This labeled data is the ground truth.

How to build a Model inspired by MNIST

Step 3: Preprocessing: MRI scans can be preprocessed by normalizing intensities and applying data augmentation techniques to simulate variations like different scanning angles or slight patient movements. This helps the model generalize better.

Step 4: Feature Extraction and Labeling: In MRI scans, key features such as the texture, intensity, and shape of regions can be extracted. Ground truth annotations involve manually labeling these features—doctors outline and classify areas of the brain as either healthy or potentially cancerous.

Step 5: Training a Simple Model: A Naive Bayes classifier can be trained on simpler, well-labeled brain scans where the distinction between tumor and non-tumor areas is clear. This model would learn the "probability distribution" of what a tumor looks like.

How to build a Model inspired by MNIST

Step 6: Incorporation of More Complex Features: features like the intensity of pixels in a region might follow a Gaussian distribution. A Gaussian Naive Bayes model can be trained to recognize tumors by learning not just which pixels are likely to belong to a tumor, but also how these pixels are distributed around a mean value.

Step 7: Handling Complex and Non-Normal Distributions: In MRI scans, the distribution of tumor features might be irregular or multimodal (having multiple peaks). KDE can model these complex distributions, providing a highly flexible tool for predicting tumor regions.

Real-World Impact of Automated Ground Truth Annotation

Efficiency: Models trained on annotated data can quickly predict tumor locations, reducing the time doctors spend on manual labeling.

Accuracy: Continuous improvement through feedback ensures that models catch even small or early-stage tumors, leading to better patient outcomes.

Scalability: These techniques can be applied to various types of medical imaging, broadening the impact across different healthcare fields.

References:

1. https://www.irjet.net/archives/V7/i6/IRJET-V7I61270.pdf
2. https://www.nature.com/articles/sdata2018158
3. https://nbia.cancerimagingarchive.net/nbia-search/?CollectionCriteria=REMBRANDT
4.

Stories

It is now story time!

Librarian

Imagine there's an ancient librarian in a vast library filled with scrolls. This librarian has seen thousands of scrolls over the years, each one labeled with its genre—history, romance, science fiction, etc. Now, without reading the entire scroll, the librarian can quickly guess the genre just by looking at a few key words.

For instance, if a scroll mentions "battle," "king," and "castle," the librarian might think it's a history scroll. If it mentions "love," "heart," and "passion," it's probably romance. The librarian doesn't care that sometimes battles appear in romance or that kings can be in science fiction; he makes a quick, simple guess based on the presence of these keywords. This is the essence of the Naive Bayes classifier—a simple, quick guess based on known associations, even if it's a bit naive!

Just like the ancient librarian, the Naive Bayes classifier makes fast predictions based on the presence of certain features. It assumes these features are independent (even if they aren't), which allows it to make quick decisions.

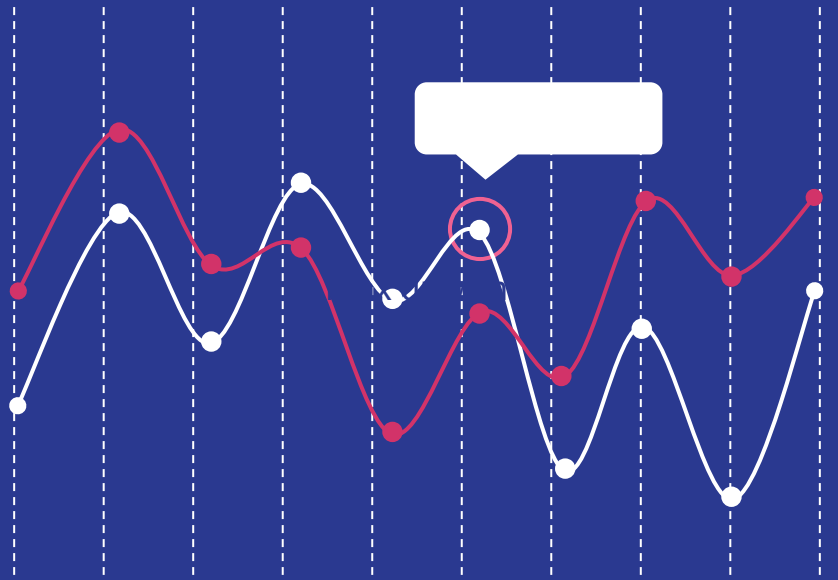
The Jelly Bean Jar

Remember those contests where you had to guess how many jelly beans are in a jar? Imagine you're really determined to win. Instead of just guessing, you decide to look at all the previous jars that people have guessed on and their actual numbers of jelly beans. But here's the twist: some jars are tall and thin, others are short and wide, and the jelly beans are all different sizes!

To make your guess, you carefully analyse the shape and size of each jar, as well as the distribution of jelly beans in all the jars you've seen before. You don't just assume that every jar has around 100 jelly beans; you estimate the likely number by smoothly blending all your past observations together, even if the jars are very different. This is what Kernel Density Estimation (KDE) does—it doesn't assume a specific pattern but uses the actual data points to make a smart guess.

The KDE classifier is like your methodical approach to the jelly bean contest. It doesn't assume a fixed pattern but rather uses the shape and spread of the data to estimate the outcome, making it more flexible and accurate for complex data.

Thank You



What is Principle Component Analysis (PCA)?

What is PCA?

Imagine you're trying to solve a big jigsaw puzzle, but the table you're working on is too small for all the pieces. You can't fit everything, so you decide to focus on the most important pieces—the ones that help you see the big picture faster, like the corners and edges. Principal Component Analysis (PCA) works in a similar way when you're dealing with large amounts of data. It helps you focus on the most important information and reduces the size of your problem, making it easier and faster to solve.

How Does PCA Work in the MNIST Model?

The Problem:

The MNIST dataset consists of images of handwritten digits, and each image has 784 pixels (28x28 pixels). That's a lot of information! When you're trying to teach a computer to recognize these digits, it has to look at all those pixels, which can be overwhelming and time-consuming.

PCA to simplify the problem:

1. Step 1: Find the Important Pixels: PCA looks at all 784 pixels and figures out which ones carry the most important information. For example, the pixels that outline the shape of the digit are more important than the pixels in the blank background.
2. Step 2: Combine the Important Information: PCA then combines these important pixels into a smaller number of "principal components." These components summarize the most essential features of the digits. Instead of dealing with 784 pieces of information, you might only need 50 or so components to capture the essence of the digit.
3. Step 3: Reduce the Dataset: By focusing on these 50 principal components instead of all 784 pixels, PCA reduces the complexity of the dataset. This means the computer can learn to recognize the digits faster and more efficiently.