VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

SCIKIT-LEARN REPORT

Topic: RECOMMENDER SYSTEM

Course: Computational Thinking

Students:	*Supervisors:*
Nguyễn Thanh Trúc (24127137)	PhD. Nguyễn Tiến Huy
Đào Minh Khoa (24127422)	PhD. Lê Thanh Tùng
Trần Lưu Gia Bảo (24127018)	Mr. Trần Hoàng Quân
Nguyễn Tiến Cường (24127337)	
Nguyễn Khánh Toàn (24127252)	
Phạm Trần Anh Quân (24127226)	

Ngày 16 tháng 11 năm 2025

Mục lục

Danh sách hình vẽ

Danh sách bảng

1 Abstract

This article list 3 predict models and compare. Dataset we use for train and test model is Fashion MNIST.
The model was evaluated on a 10-class FashionMNIST dataset including categories such as T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

2 Introduction

In this report, we present and compare three widely used machine learning algorithms for classification tasks: K-Nearest Neighbors (KNN), Naive Bayes, and Decision Tree. Each of these algorithms has its own strengths and weaknesses, making them suitable for different types of datasets and applications.

3 Decision Tree

3.1 Definition

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks.
It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.
For multiclass classification, the model predicts one class among multiple possible categories by recursively splitting the input space based on the most informative features.
Each internal node represents a test on a feature, each branch represents an outcome of that test, and each leaf node represents a final decision or prediction.

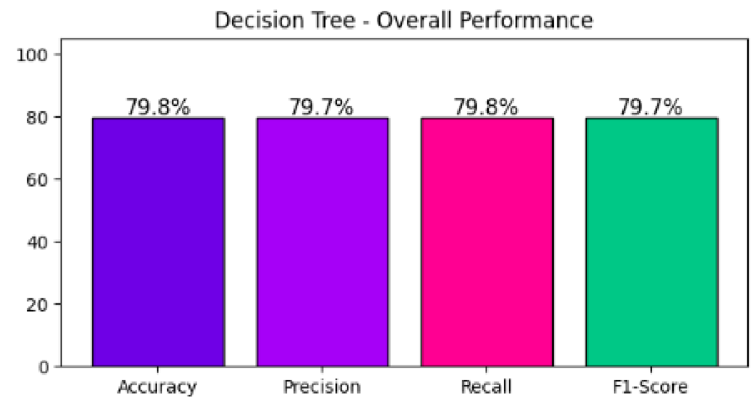
3.2 Working Principle

Decision tree learning employs a divide and conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels.

Start

1. Start with the entire dataset at the root node.
2. Evaluate all features and choose the one that best separates the data.
3. Split the dataset based on this feature.
4. Repeat the process recursively for each child node until:
   - All samples belong to the same class, or
   - Maximum tree depth is reached, or
   - No further improvement can be made.

3.3 Overall Performance



Hình 1: Decision Tree Performance

These results show that the Decision Tree provides moderate performance for this multiclass task, with balanced precision and recall.

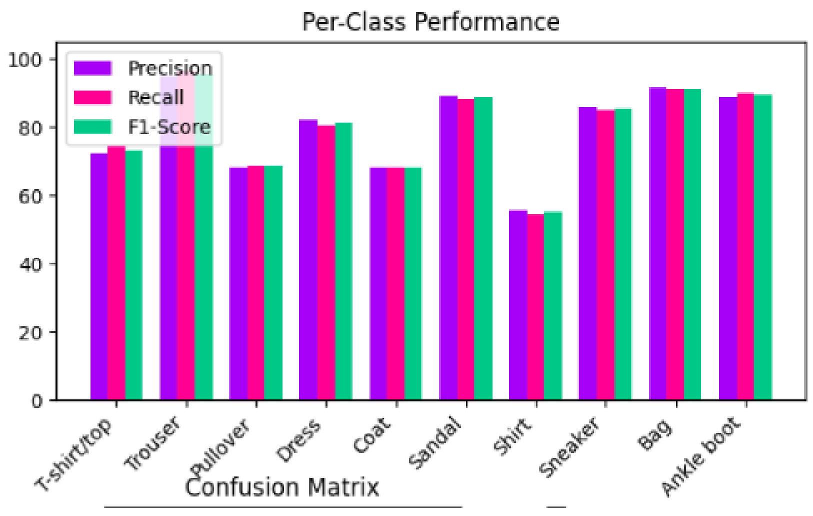
3.4 Class Performance

3.5 Accuracy per class

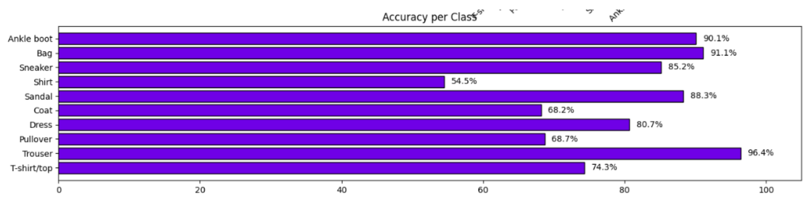
Observations:

Trouser, Bag, Ankle boot are classified extremely well ($> 90\%$)

Shirt has the lowest accuracy ($\approx 54\%$), a common challenge due to visual overlap with T-shirt or Coat.



Hình 2: Decision Tree Per-class Performance



Hình 3: Decision Tree Per-class Accuracy

3.6 Advantages and Limitations

Advantages:

1. **Interpretable structure**

   The model provides clear decision rules, allowing easy interpretation of how the 10 clothing categories are classified.

2. **Fast training time**

   Training is computationally inexpensive, even with 60,000 images, making it suitable for quick baseline modeling.

3. **Good performance on some distinct classes**

   Classes with simple visual patterns (e.g., Trouser, Bag, Ankle boot) achieved high accuracy ($> 90\%$), showing that the tree can separate well-defined categories.

4. **Non-linear decision boundaries**

   The tree can model non-linear relationships between pixel features without requiring feature scaling.

Limitations:

1. **Relatively low overall accuracy** (79.8%)

   For high-dimensional image data, a single decision tree is not strong enough to capture complex patterns compared to modern models (CNNs, Random Forest, XGBoost).

2. **High sensitivity to noise and similar classes**

   Classes with visually overlapping shapes (Shirt, Pullover, Coat) show poor accuracy (as low as $\approx 54\%$), indicating instability in boundary decisions.

3. **Overfitting tendency**

   Without proper pruning, the model memorizes training data and generalizes poorly on test samples.

4. **Poor handling of continuous, high-dimensional pixel inputs**

   FashionMNIST images (784 features) lead to fragmented splits and reduced discriminative power.

5. **Unbalanced class performance**

   Large performance variance between classes ($54\% \rightarrow 96\%$) suggests the model is not stable across all categories.

4 K-Nearest Neighbors (KNN)

4.1 Definition

K-Nearest Neighbors (KNN) is a simple, intuitive, yet powerful Machine Learning algorithm used for classification problems. It is considered one of the most understandable algorithms in Machine Learning and is especially suitable for beginners. [2]

The algorithm is based on the assumption that similar data points are close to each other, while different data points are farther apart in the feature space.

4.2 Working Principle

4.2.1 Classification Process

KNN classifies a new data point using the following steps:

Step 1: Distance Calculation

Compute the distance from the new data point to all labeled data points in the training set. Common distance types include:

- Euclidean Distance: Straight-line distance in space.
- Manhattan Distance: Distance along perpendicular grid-like paths.
- Cosine Distance: Often used for text or similarity-based problems.

Step 2: Determine K Nearest Neighbors

Select K labeled data points having the shortest distances to the target point.

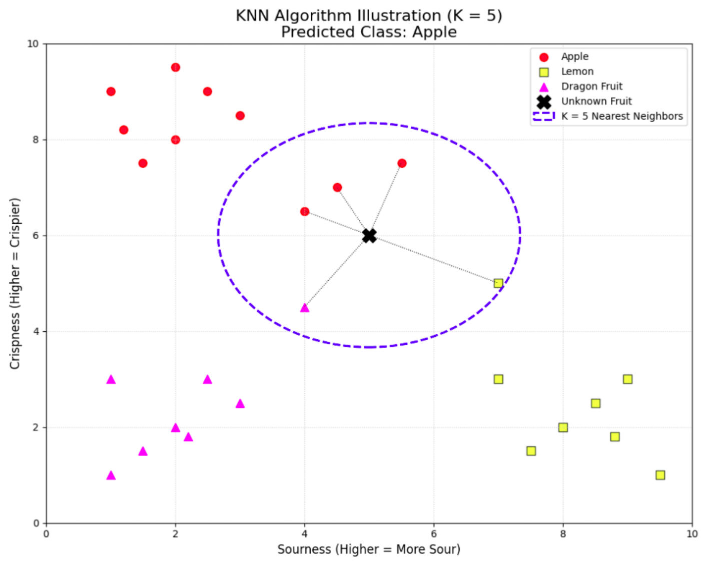
Typical K values: 1, 3, 5, 7,etc. **Step 3: Majority Voting**

Count how many of the K neighbors belong to each class.

The class that appears the most is assigned to the new data point.

4.2.2 Illustrative Example

Imagine a 2D chart plotting known fruits using:



Hình 4: KNN Classification Example

- X-axis: Sourness (higher → more sour)
- Y-axis: Crispness (higher → crispier)

After plotting the known fruits:

- Apples (red dots): High crispness, low sourness

- Lemons (yellow squares): Low crispness, high sourness
- Dragon fruit (purple triangles): Low crispness, low sourness

Problem:
We have an Unknown Fruit (?), located around the center (medium sourness, medium crispness).

KNN Classification Steps:

- Choose K = 5
- Find the 5 nearest neighbors
- Suppose the neighbors include:
  - 3 Apples
  - 1 Lemon
  - 1 Dragon fruit

Conclusion:
The unknown fruit is predicted to be an Apple because the majority of its neighbors belong to the Apple class.

4.2.3 Factors Affecting Performance

Selection of K:

- K too small $\rightarrow$ Overfitting
- K too large $\rightarrow$ Important local patterns may be lost

Selection of distance type:

- Must match the nature of the dataset
- Requires experimentation to find the best fit

4.2.4 Parameter Optimization

To find the best K and distance metric:

1. Split data into training and test sets

2. Try different distance types

3. Try different values of K

4. Measure accuracy

5. Choose the highest-performing combination

4.2.5 Data Characteristics

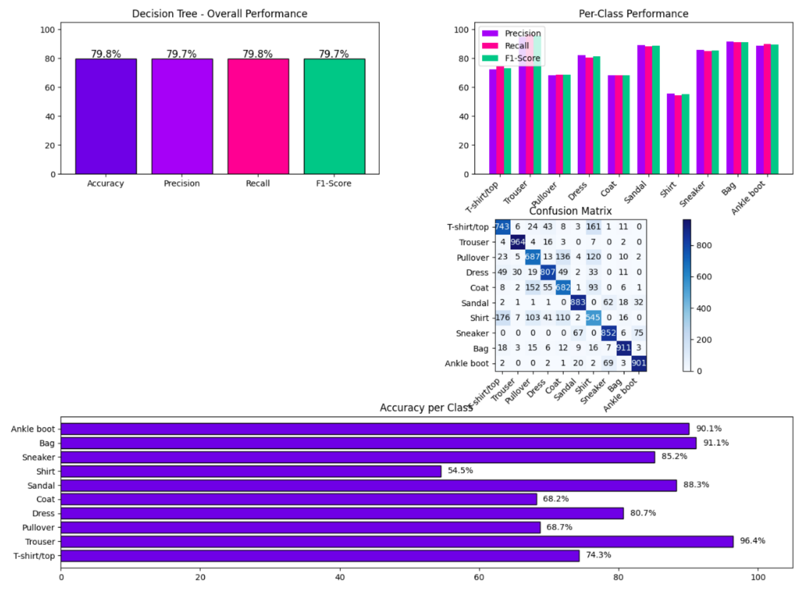
KNN works best when:

- Data is clearly separable by distance
- Each class forms compact clusters
- Variance within classes is low

When data becomes more dispersed or overlapping, performance decreases.

4.3 Performance Result

4.3.1 Visualization



Hình 5: KNN Performance by: Overall Metrics, Per-Class Performance, Confusion Matrix, and Accuracy by Class.

4.3.2 Observations

(a) **Overall Performance** The model achieved strong performance on the test dataset with the following macro-averaged metrics:

Accuracy: 85.73%

Precision: 85.99%

Recall: 85.73%

F1-Score: 85.63%

These metrics are shown in the "Overall Performance Metrics" chart.

(b) **Detailed Analysis Through Visualizations Accuracy by Class**

The “Accuracy by Class” horizontal bar chart shows significant variation across categories. High-performing classes:

- Trouser: 96.5%
- Ankle Boot: 96.2%
- Bag: 94.9%
- Sneaker: 94.9%

These items have distinctive shapes, making them easier to classify.

Low-performing class:

- Shirt: 57.7%, the lowest accuracy.

Confusion Matrix Insights The confusion matrix explains why Shirt performs poorly:

- Misclassified as T-shirt/top: 205 times
- Misclassified as Pullover: 115 times
- Misclassified as Coat: 74 times

Shirt is frequently confused with other upper-body clothing categories, which share similar visual features. Classes like Trouser, Sneaker, and Bag show high diagonal values (e.g., 965, 949, 952), meaning they are rarely misclassified. **Per-Class Performance Metrics**

The “Per-Class Performance” chart further confirms this:

Shirt has very low Recall ($\approx$ 60%).

This aligns with its 57.7% accuracy.

4.3.3 Conclusion

The KNN model with k = 10 achieves an overall accuracy of 85.73% on Fashion MNIST. It performs very well for categories with distinctive shapes such as trousers, bags, and shoes.
However, the model struggles with categories having overlapping visual features, particularly Shirt, which is often confused with T-shirt/top, Pullover, and Coat. This highlights a core limitation of KNN when class boundaries are not well separated in distance space.

5 Naive Bayes

5.1 Definition

Naive Bayes is a straightforward yet effective Machine Learning algorithm used for classification tasks. It's built upon Bayes' Theorem, a fundamental concept in probability.
The algorithm is called "naive" because it makes a key assumption: it believes that all data features are independent of each other, given the class of the item. While this assumption is rarely perfectly true in real-world scenarios (especially with images), Naive Bayes often performs surprisingly well, particularly in tasks like text classification.[1]

5.2 Working Principle

5.2.1 Classification Process

Unlike KNN, which measures distances, Naive Bayes works by calculating probabilities.
Step 1: Probability Calculation (Training Phase)
The model "learns" from the training data by computing two main types of probabilities:

- Prior Probability: This is the probability of each class appearing in the dataset. For example, P(T-shirt/top) is the proportion of T-shirt/top images among all 60,000 training images.
- Conditional Probability: This is the probability of a specific feature (like a pixel's brightness) occurring, given that we already know the item's class. For example, $P(Pixel_{100} = Bright|T-shirt/top)$ tells us how likely a specific pixel at position 100 is to be bright if the image is indeed a T-shirt/top.

Step 2: Posterior Probability Calculation (Prediction Phase)
When a new data point (a new image) arrives, the model uses Bayes' Theorem to calculate the

probability that it belongs to each possible class:

$$P(ClassA|NewImage) \alpha P(NewImage|ClassA) \times P(ClassA)$$

This means the probability that the new image is of Class A is proportional to the probabilities of seeing that image given Class A, multiplied by the prior probability of Class A.
The "naive" assumption simplifies $P(NewImage|ClassA)$ into a product of individual pixel probabilities:

$$P(Pixel_1|ClassA) \times P(Pixel_2|ClassA) \times \dots$$

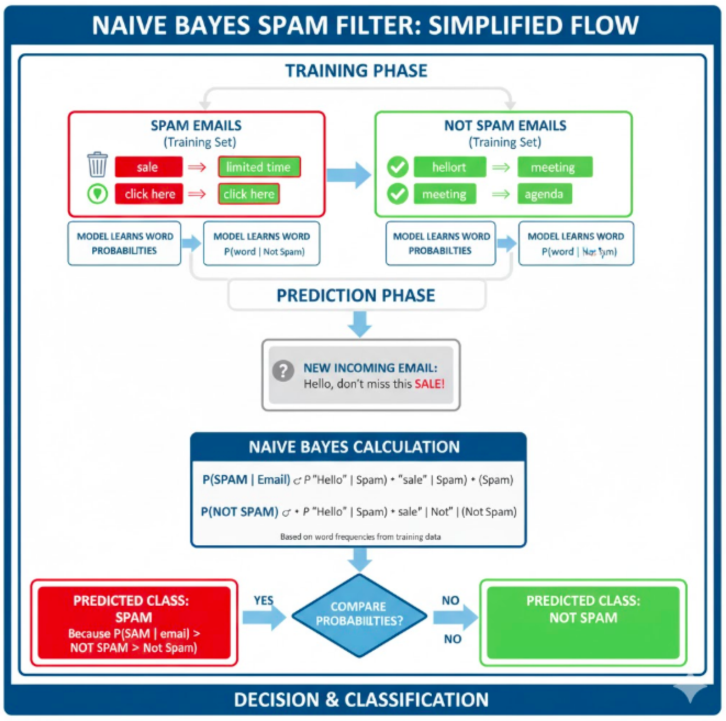
This means it treats each pixel as independent, ignoring how they might relate to each other to form shapes.

Step 3: Decision (Classification)

The class with the highest posterior probability is then assigned to the new data point.

5.2.2 Illustrative Example

Imagine we are building a spam email classifier using Naive Bayes.


NAIVE BAYES SPAM FILTER: SIMPLIFIED FLOW
TRAINING PHASE
SPAM EMAILS (Training Set)
sale
limited time
click here
click here
NOT SPAM EMAILS (Training Set)
hellort
meeting
meeting
agenda
MODEL LEARNS WORD PROBABILITIES
MODEL LEARNS WORD P(word | Not Spam)
MODEL LEARNS WORD PROBABILTIES
MODEL LEARNS WORD P(word | Not Spam)
PREDICTION PHASE
NEW INCOMING EMAIL: Hello, don't miss this SALE!
NAIVE BAYES CALCULATION
P(SPAM | Email) ∝ P "Hello" | Spam) • "sale" | Spam) • (Spam)
P(NOT SPAM) ∝ • P "Hello" | Spam) • sale" | Not" | (Not Spam)
Based on word frequencies from training data
PREDICTED CLASS: SPAM Because P(SAM | email) > NOT SPAM > Not Spam)
YES
COMPARE PROBABILTIES?
NO
NO
PREDICTED CLASS: NOT SPAM
DECISION & CLASSIFICATION

Hình 6: Naive Bayes Classification Example

- Features: Words found in an email (e.g., "sale", "report", "hello").

- Classes: "Spam" or "Not Spam".

Problem:
A new email arrives with the subject: "Hello, don't miss this SALE!"
Naive Bayes Classification Steps:

1. Learning from Data (Training): The model first analyzes a large set of known "Spam" and "Not Spam" emails. It counts how often each word appears in each category. From this, it learns:
    - The overall likelihood of an email being Spam vs. Not Spam (P(Spam) or P(Not Spam)).
    - The likelihood of a specific word appearing given that the email is Spam or Not Spam (P("word" | Spam) or P("word" | Not Spam)). For example, P("sale" | Spam) would likely be high.
2. Predicting a New Email: For the new email "Hello, don't miss this SALE!":
    - The model calculates P(Spam | "Hello", "sale") and P(Not Spam | "Hello", "sale").
    - Crucially, due to the "naive" assumption, it simplifies these calculations. For instance: $P(Spam|email) \alpha P(”Hello”|Spam) * P(”sale”|Spam) * P(Spam)$
    - It does the same for "Not Spam".
3. Decision: It compares these two calculated probabilities.

Conclusion:

- If P(Spam | email) is greater than P(Not Spam | email), the email is predicted as "Spam".
- Otherwise, it's classified as "Not Spam".

This process allows Naive Bayes to classify new emails based on the statistical likelihood of its words belonging to either category.

5.2.3 Factors Affecting Performance

extbfIndependence assumption: The central assumption behind Naive Bayes is conditional independence of features given the class. This is the single most important factor affecting performance. When features are highly correlated (for example, adjacent pixels in an image that jointly form shapes), the independence assumption is violated and Naive Bayes' predictive accuracy degrades. In domains where features interact in complex ways, the model cannot capture those dependencies. extbfTypes of Naive Bayes: Different Naive Bayes variants are tailored to different data types. Choosing the correct variant is crucial:

- **GaussianNB:** Assumes continuous features follow a Gaussian distribution (suitable for real-valued inputs).
- **MultinomialNB:** Designed for count data (e.g., word counts in text); models the distribution of term frequencies.
- **BernoulliNB:** Appropriate for binary/boolean features (presence/absence of a token or attribute).

extbfData preprocessing: Because Naive Bayes relies on feature-wise probability estimates and independence, careful preprocessing can improve performance:

- *Scaling/Normalization:* For GaussianNB, standardizing features (zero mean, unit variance) can make the Gaussian assumption more plausible.
- *Dimensionality reduction (PCA):* Principal Component Analysis can create new orthogonal features (principal components) that are less correlated, which better satisfies the independence assumption and often improves accuracy on image or dense numerical data.
- *Feature selection / engineering:* Remove noisy or redundant features, and create informative aggregated features (e.g., counts, ratios) where appropriate.

5.2.4 Parameter Optimization

To find the best-performing Naive Bayes configuration:

1. Experiment with preprocessing: apply StandardScaler (for GaussianNB) and/or PCA to reduce dimensions and decorrelate features.
2. Vary PCA `n_components` to control how many principal components to retain; evaluate downstream accuracy.
3. For `GaussianNB`, tune `var_smoothing` (a small constant added to variances) to prevent zero-variance issues.

4. Use cross-validation with GridSearchCV (or RandomizedSearchCV) to search combinations of preprocessing steps and model hyperparameters and select the configuration with the best validation performance.

5.2.5 Data Characteristics

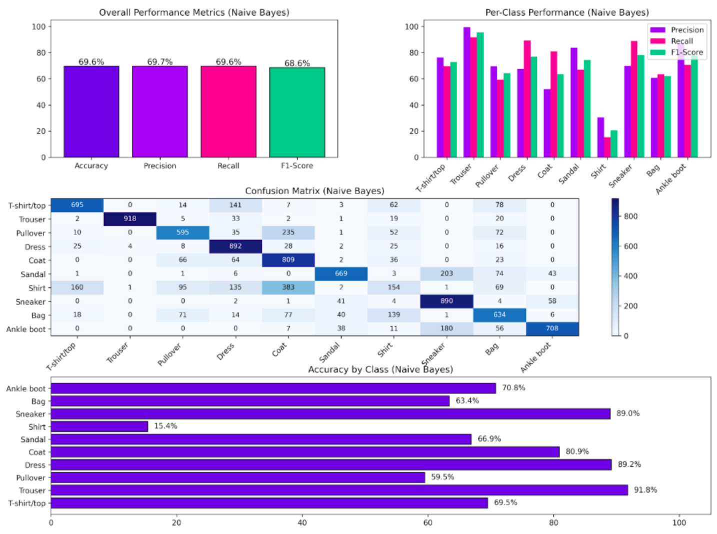
Naive Bayes generally performs well when:

- Features are approximately independent given the class.
- The dataset has high dimensionality (e.g., text data with many word features), where simple probabilistic assumptions can be effective.
- The training set is reasonably large so that reliable per-feature probability estimates can be obtained.

It performs poorly when features exhibit complex, interdependent relationships (e.g., pixels forming intricate visual patterns), in which case discriminative or deep models (SVMs, CNNs) that model feature interactions tend to outperform Naive Bayes.

5.3 Performance Result

5.3.1 Visualization



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	695	0	14	141	7	3	62	0	78	0
Trouser	2	918	5	33	2	1	19	0	20	0
Pullover	10	0	595	35	235	1	52	0	72	0
Dress	25	4	8	892	28	2	25	0	16	0
Coat	0	0	66	64	809	2	36	0	23	0
Sandal	1	0	1	6	0	669	3	203	74	43
Shirt	160	1	95	135	383	2	154	1	69	0
Sneaker	0	0	0	2	1	41	4	890	4	58
Bag	18	0	71	14	77	40	139	1	634	6
Ankle boot	0	0	0	0	7	38	11	180	56	708

Hình 7: Naive Performance by: Overall Metrics, Per-Class Performance, Confusion Matrix, and Accuracy by Class.

5.3.2 Observations

(a) **Overall Performance** The model achieved moderate performance on the test dataset. The macro-averaged metrics (calculated across all classes) are as follows:
Accuracy: 69.6%
Precision: 69.7%
Recall: 69.6%
F1-Score: 68.6%
These key metrics are visually summarized in the "Overall Performance Metrics" chart (top-left panel of the dashboard).

(b) **Detailed Analysis Through Visualizations** extbfAccuracy by Class: The horizontal bar chart titled "Accuracy by Class" reveals significant variation in performance across categories:

- *High-performing classes:*
  - Trouser: 91.8%
  - Dress: 89.5%
  - Sneaker: 89.0%

  These items tend to have distinctive shapes and fewer visual overlaps with other categories, making them easier for the model to classify.
- *Lowest-performing class:*
  - Shirt: 15.4%, which is notably the lowest accuracy among all classes.

extbfConfusion Matrix Insights: The confusion matrix provides a crucial explanation for the poor performance of the "Shirt" class:

- Out of 1,000 actual "Shirt" images in the test set, the model correctly predicted 383 of them (diagonal count for Shirt). The overall Recall for Shirt (15.4%) indicates a low true-positive rate relative to all Shirt instances.
- The model frequently misclassifies "Shirt" into other upper-body apparel categories:
  - Misclassified as T-shirt/top: 160 times
  - Misclassified as Pullover: 95 times
  - Misclassified as Coat: 135 times

This highlights that the Naive Bayes model, even when combined with PCA, struggles to distinguish subtle features (like collars or buttons) that differentiate various types of shirts and coats.

extbfPer-Class Performance Metrics: The "Per-Class Performance" bar chart further confirms these observations: The "Shirt" class exhibits extremely low Recall (represented by the pink bar, #ec4899) and F1-Score (green bar, #10b981), indicating very poor classification performance compared to other classes.

5.3.3 Conclusion

The Naive Bayes model, even after extensive optimization using PCA for dimensionality reduction and GridSearchCV for hyperparameter tuning, achieved an overall accuracy of 69.6
The model demonstrates strong performance for categories with distinct geometric shapes, such as trousers and shoes.
However, it significantly struggles with categories that share overlapping visual characteristics—most notably the "Shirt" class. "Shirt" is frequently confused with other upper-body garments like "T-shirt/top", "pullover", and "coat". This outcome underscores the fundamental limitation of the "naive" assumption: it is inherently ill-suited for understanding the complex spatial relationships of pixels within image data.

Tài liệu

[1] CampusX. Naive bayes classifier | part 1 | conditional probability, 2020.

[2] Học Lập Trình cùng Phát. Machine learning cơ bản: Thuật toán k nearest neighbors (knn) trong 10 phÚt, 2025.