# Introduction to Data Science

-K-Nearest Neighbors & Naive Bayes Classification -

Mauricio Molina

Keio University, Faculty of Economics

Jun 25, 2025

## Contents

1. K-Nearest Neighbors
2. Confusion Matrix & Performance Metrics
3. Naive Bayes Classifiers
4. Classifier Comparison

## What Is K-Nearest Neighbors (KNN)?

Predicting qualitative responses is a process that is known as **classification**. Predicting a qualitative response for an observation can be referred to as classifying classification that observation, since it involves assigning the observation to a category, or class.

The **K-Nearest Neighbors** (KNN) method, for example, is quite simple: classify a record in accordance with how similar records are classified.

## K-Nearest Neighbors

For each record to be classified or predicted:

1. Find K records that have similar features (i.e., similar predictor values).
2. For classification, find out what the majority class is among those similar records and assign that class to the new record.
3. For prediction (also called KNN regression), find the average among those similar records, and predict that average for the new record.

KNN is one of the simpler prediction/classification techniques: there is no model to be fit. The prediction results depend on how the features are scaled, how similarity is measured, and how big K is set. Also, all predictors must be in numeric form. We will illustrate how to use the KNN method with a classification example.

## K-Nearest Neighbors (KNN) Classifier

From a variable-sized dataset, $\mathcal{D} = \{(x_n, y_n) : n = 1, \ldots, N\}$:

- The KNN classifier assigns a new input $\mathbf{x}$ to the class most common among its $K$ nearest neighbors in the training set $\mathcal{D}$.
- Let $\mathcal{N}_K(\mathbf{x}, \mathcal{D})$ denote the set of the $K$ closest examples to $\mathbf{x}$ in $\mathcal{D}$. Then the class-conditional distribution is

$$p(y = c \mid \mathbf{x}, \mathcal{D}) = \frac{1}{K} \sum_{\mathbf{x}_n \in \mathcal{N}_K(\mathbf{x}, \mathcal{D})} \mathbf{I}(y_n = c).$$

- The two main hyperparameters are:
  1. The neighborhood size $K$.
  2. The distance metric $d(\mathbf{x}, \mathbf{x}')$.
- **Mahalanobis distance:**

$$d_M(\mathbf{x}, \boldsymbol{\mu}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^\mathsf{T} M (\mathbf{x} - \boldsymbol{\mu})},$$

  where $M$ is any positive-definite matrix. If $M = I$, this reduces to the Euclidean distance.

## K-Nearest Neighbors in 2D (K = 5)

- For a new test point $\mathbf{x}$, compute distances to all training points.
- Identify the $K = 5$ nearest neighbors in feature space.
- Count the labels of those 5 neighbors: here they are $\{1, 1, 1, 0, 0\}$.
- Assign $\mathbf{x}$ to the majority label $\rightarrow$ class "1."
- Hence, we predict $p(y = 1|\mathbf{x}, \mathcal{D}) = 3/5 = 0.6$

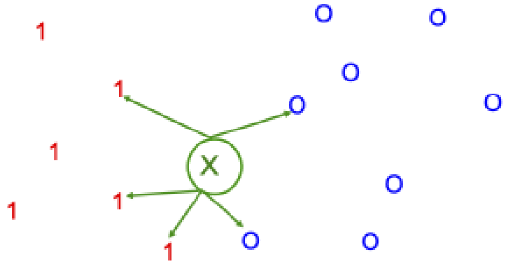


Illustration of a K-nearest neighbors classifier in 2D for $K = 5$. The nearest neighbors of test point $\mathbf{x}$ have labels $\{1, 1, 1, 0, 0\}$.

## Voronoi Tessellation for 1-NN

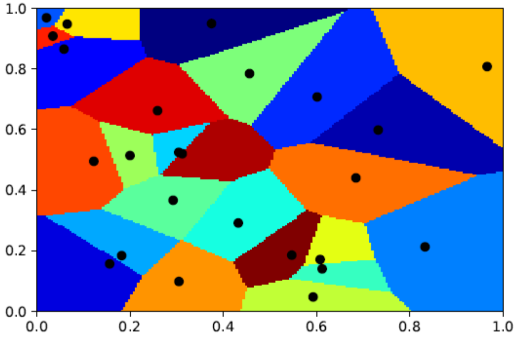


Illustration of the Voronoi tessellation induced by 1-NN.

- When $K = 1$, each training point "owns" the region of the plane where it is the nearest neighbor.
- The resulting partition of feature space is called the Voronoi tessellation.
- A new point is classified according to the single nearest training point.
- Voronoi cells are convex polygons in 2D (generalized to polyhedra in higher dimensions).

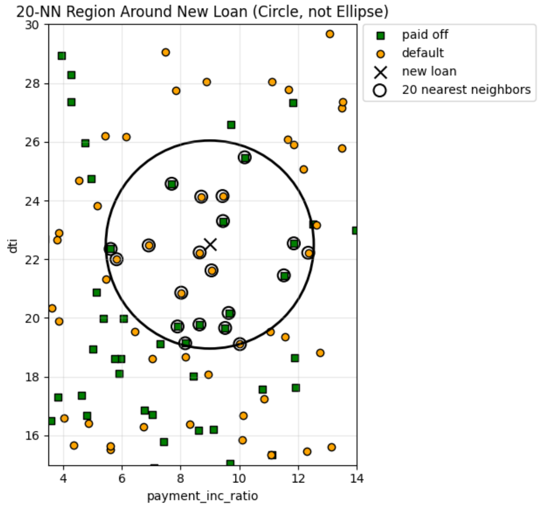
## Example on Lending Club data

- **Dataset:** “loan200” consists of 200 LendingClub personal loans
  - *LendingClub* is a peer-to-peer lending platform where investors fund individual loans
  - Each loan has a known binary outcome: “paid off” vs. “default” (variable `outcome200`)
- **Predictors:**
  - $\texttt{dti}$ = Debt-to-Income ratio (excluding mortgage), scaled by 100
  - $\texttt{payment\_inc\_ratio}$ = Loan payment ÷ Income, scaled by 100
- **Goal:** Given a new loan with

$$\texttt{dti} = 22.5, \quad \texttt{payment\_inc\_ratio} = 9.0,$$

  use KNN (with $K = 20$) to predict whether it will “paid off” or “default.”
- **Procedure:**
  1. Compute distances from the new loan $\left(22.5,\ 9.0\right)$ to all 200 known loans in the 2D feature space
  2. Identify the 20 nearest neighbors in that feature space
  3. Assign the new loan to the majority class among those 20 neighbors

# KNN Output & Probability Interpretation



- The large black circle shows the boundary of the nearest 20 points. In this case, 9 defaulted loans lie within the circle, as compared with 11 paid-off loans. Hence the predicted outcome of the loan is paid off. Note that if we consider only three nearest neighbors, the prediction would be that the loan defaults.
- Although KNN classification typically returns a binary label ("default" vs. "paid off"), most implementations can also output a *probability* (propensity) for each class.
- The probability of "default" is simply: number of neighbors labeled "default" divided by K.
- In this example, suppose among the 20 nearest neighbors, 9 have label "default." Then

$$\hat{P}(\text{default} \mid \text{new loan}) = \frac{9}{20} = 0.45.$$

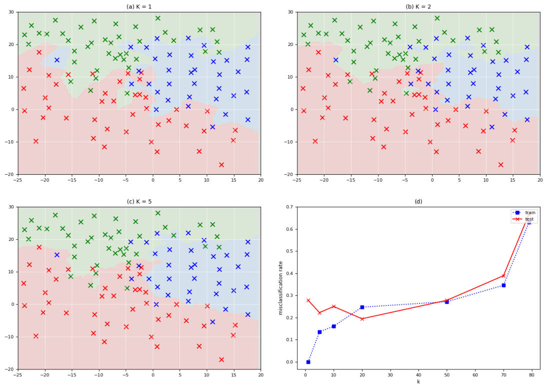


Figure: Decision boundaries induced by a KNN classifier. (a)$K = 1$. (b) $K = 2$. (c) $K = 5$. (d) Train and test error vs $K$.

As K increases, the decision boundaries become smoother (since we are averaging over larger neighborhoods), so the training error increases, as we start to underfit.

## The Curse of Dimensionality

- **Main issue:** KNN classifiers break down in high dimensions due to the *curse of dimensionality*.
- **Why?**
  - In $D$ dimensions, volume grows exponentially fast.
  - To capture a fixed fraction $p$ of the data around a test point $\mathbf{x}$, a hypercube must be very large.
- **Example:**
  1. Inputs $\mathbf{x}$ are uniformly distributed in $[0,1]^D$.
  2. Grow a hypercube centered at $\mathbf{x}$ until it contains fraction $p$ of points.
  3. The expected edge length is
$$e_D(p) = p^{1/D}.$$
- **Implications for $D = 10$:**
  - If $p = 0.10$, then $e_{10}(0.10) = 0.10^{1/10} \approx 0.80$.
  - If $p = 0.01$, then $e_{10}(0.01) = 0.01^{1/10} \approx 0.63$.
  - Since each coordinate runs from 0 to 1, a cube of side 0.80 or 0.63 is huge.
- **Key Takeaway:** "Nearest" neighbors lie far away along most dimensions $\rightarrow$ poor local estimates.

**Why Far-Away Neighbors Hurt**

- Large hypercubes to capture small $p \Rightarrow$ averaging over a wide region.
- Neighbors may differ drastically in many coordinates.
- Locality vanishes, so KNN no longer "local."

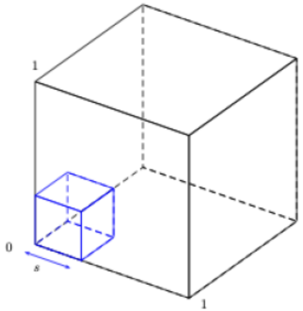
**Two Principal Remedies**

- **Use a Parametric Model**
  - Impose global structure (e.g., linear or logistic regression, decision trees).
  - Reduces reliance on exponentially large neighborhoods.
- **Dimension-Selective Metrics**
  - *Feature Selection:* restrict KNN to a smaller subset of coordinates.
  - *Metric Learning / Mahalanobis:* weight/transform dimensions so that "closeness" reflects relevance.
  - Effective dimensionality is lowered, neighbors stay truly local.

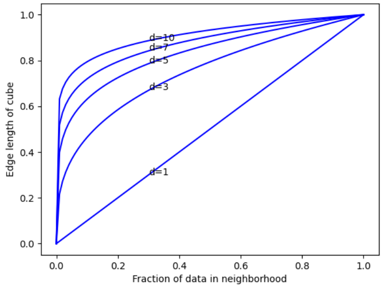
**Summary:**

- In low $D$, KNN is genuinely local.
- As $D$ increases, required edge length $e_D(p) = p^{1/D}$ becomes large.
- Remedy by adding structure (parametric) or reducing/weighting dimensions (metric learning or feature selection).

## Curse of Dimensionality



(a) Embedding a small cube of side $s$ inside the unit cube



(b) Edge length $e_D(p) = p^{1/D}$ vs. dimension $D$

(a) A small cube of side $s$ inside a larger unit cube. (b) Plot of $e_D(p)$ vs. $D$, showing how far the cube must extend in each dimension to capture fraction $p$.

## Choosing $K$ in KNN

- **Importance of $K$:**
  - $K = 1$ is the 1-nearest neighbor classifier: predict based on the single closest training point.
  - In practice, $K = 1$ often overfits; better performance usually achieved with $K > 1$.
- **Effects of $K$:**

  Small $K$ (e.g., 1) $\rightarrow$ High variance, prone to overfitting and noise.

  Large $K$ $\rightarrow$ Smoother decision boundary, reduced variance, but risk of *oversmoothing*.
- **Choosing the "best" $K$:**
  - No universal rule—depends on data structure and noise level (signal-to-noise ratio).
  - Use cross-validation or a held-out validation set to compare accuracy for different $K$.
  - Typical range: $K \approx 1$ to 20.
  - Often pick an *odd* $K$ to avoid ties when classes are binary.

## Understanding Confusion Matrix

### Evaluating Classification Performance

**What is a Confusion Matrix?**

A table that visualizes classification model performance by comparing predicted vs actual labels. Essential for binary and multi-class classification.

**Binary Classification Matrix:**

| | | Actual | |
|---|---|---|---|
| | | **Positive** | **Negative** |
| **Predicted** | **Positive** | TP | FP |
| | **Negative** | FN | TN |

**Key Components:**

- **TP (True Positive):** Correct positive predictions
- **FP (False Positive):** Negative cases misclassified as positive (Type I error)
- **TN (True Negative):** Correct negative predictions
- **FN (False Negative):** Positive cases misclassified as negative (Type II error)

**Performance Metrics:**

- **Accuracy:** $\frac{TP+TN}{TP+TN+FP+FN}$ (Overall correctness)
- **Precision:** $\frac{TP}{TP+FP}$ (How accurate positive predictions are)
- **Recall/Sensitivity:** $\frac{TP}{TP+FN}$ (Ability to detect positives)
- **F1-score:** $2 \times \frac{Precision \times Recall}{Precision + Recall}$ (Harmonic mean of precision/recall)
- **Specificity:** $\frac{TN}{TN+FP}$ (Ability to detect negatives)

## Bias–Variance Trade-off in KNN

**Definitions:**

- **Variance:** Error due to variation in *training data*. Different training sets yield different models.
- **Bias:** Error due to model's inability to capture the true underlying structure. Persistence of error even with infinite data.

**Trade-off in KNN**

- $K$ is too Small:
  - Model is very flexible $\rightarrow$ low bias.
  - High sensitivity to noise $\rightarrow$ high variance.
- $K$ is too Large:
  - Model is smoother $\rightarrow$ low variance.
  - Decision boundary might miss local structure $\rightarrow$ high bias.
- The "optimal" $K$ balances bias and variance to minimize total error.

## Naive Bayes: Overview

- The naive Bayes algorithm uses *conditional probabilities* $P(X_j \mid Y = k)$ to estimate the *posterior probability* $P(Y = k \mid X_1, \dots, X_p)$.
- In other words, we want
$$P(Y = k \mid X_1, \dots, X_p)$$
but we start from
$$P(X_1, \dots, X_p \mid Y = k) \quad \text{and} \quad P(Y = k).$$
- **Key terms:**
  - *Conditional probability:* $P(X = x \mid Y = k)$ is the probability that predictor $X$ takes value $x$ given class $Y = k$.
  - *Posterior probability:* $P(Y = k \mid X_1, \dots, X_p)$ is the probability of outcome $Y = k$ *after* observing predictors. It contrasts with the *prior* $P(Y = k)$, which ignores predictor values.

- **Exact Bayesian classification (conceptual):**
  1. For a new record with predictors $(X_1, \ldots, X_p)$, find all training records that match exactly on every predictor.
  2. Among those exact matches, determine which class label is most prevalent.
  3. Assign that class to the new record.
- This exact approach is like computing

$$P(Y = k \mid X_1, \ldots, X_p) \; = \; \frac{P(Y = k)\, P(X_1, \ldots, X_p \mid Y = k)}{\sum_{i=1}^{K} P(Y = i)\, P(X_1, \ldots, X_p \mid Y = i)},$$

  but only using records with $(X_1, \ldots, X_p)$ exactly equal.

**Problem:** As the number of categorical predictors increases, the likelihood of finding any training record that matches a new record on *all* predictors goes to zero.

## Why Exact Bayesian Classification Fails in Practice

- *Example:*
  - Predict voting by gender, ethnicity, income bracket, region, voting history, number of children, marital status, etc.
  - Even a large sample may not contain a single record matching a new record who is, say, male, Hispanic, high-income, Midwest, voted last election, did not vote prior election, three daughters, one son, divorced.
  - That is already eight predictors—typical classification problems have many more.
- Adding one more predictor with 5 equally common categories reduces the probability of an exact match by a factor of 5.
- Consequently, most new records have *zero* exact matches $\rightarrow$ cannot estimate $P(X_1, \ldots, X_p \mid Y = k)$ by direct counting.

- **Naive Bayes remedy:** Use the *entire dataset* but *assume conditional independence*:

$$P(X_1, \ldots, X_p \mid Y = k) \approx \prod_{j=1}^{p} P(X_j \mid Y = k).$$

- This "naive" assumption lets us compute posterior probabilities even when exact matches do not exist.

Using this "naive" assumption, the posterior probability can be computed as:

$$P(Y = k|X_1, \dots, X_p) = \frac{P(Y = k) \prod_{j=1}^{p} P(X_j|Y = k)}{\sum_{i=1}^{K} P(Y = i) \prod_{j=1}^{p} P(X_j|Y = i)}$$

We still need to specify the form of the probability distributions for $X_j|Y = k$. This depends on what type of feature $X_d$ is.

## Continuous Variables: Gaussian Naïve Bayes

- **Assumption:** Each continuous feature $X_j \mid Y = k$ follows $\mathcal{N}(\mu_{j,k}, \sigma^2_{j,k})$.
- Likelihood per feature:

$$f(x_j \mid Y = k) = \frac{1}{\sqrt{2\pi}\,\sigma_{j,k}} \exp\Big(-\frac{(x_j - \mu_{j,k})^2}{2\sigma^2_{j,k}}\Big)$$

- Compute parameters $\mu_{j,k}$ and $\sigma_{j,k}$ from training data:

$$\mu_{j,k} = \frac{1}{N_k} \sum_{i:y_i=k} x_{ij}, \quad \sigma^2_{j,k} = \frac{1}{N_k} \sum_{i:y_i=k} (x_{ij} - \mu_{j,k})^2$$

- Prior $P(Y = k) = \frac{N_k}{N}$.
- Posterior score (log form to avoid underflow):

## Binary Features: Bernoulli Naïve Bayes

- **Use case:** Binary features (e.g., presence/absence of words in documents, true/false attributes)
- **Assumption:** Each feature $X_j \mid Y = k$ follows Bernoulli distribution:

$$P(X_j = x_j \mid Y = k) = \begin{cases} \theta_{j,k} & \text{if } x_j = 1 \\ 1 - \theta_{j,k} & \text{if } x_j = 0 \end{cases}$$

- Likelihood for feature vector:

$$P(\mathbf{X} \mid Y = k) = \prod_{j=1}^{p} \theta_{j,k}^{x_j} (1 - \theta_{j,k})^{1-x_j}$$

- Parameter estimation ($\theta_{j,k}$ = probability of feature $j$ being present in class $k$):

$$\theta_{j,k} = \frac{N_{j,k}}{N_k}$$

where $N_{j,k}$ = number of class $k$ samples with $X_j = 1$, $N_k$ = total class $k$ samples

## Bernoulli NB: Handling and Smoothing

- Explicitly models absence of features ($X_j = 0$)
- Posterior probability:

$$P(Y = k \mid \mathbf{X}) \propto P(Y = k) \prod_{j=1}^{p} \theta_{j,k}^{x_j} (1 - \theta_{j,k})^{1-x_j}$$

- **Laplace smoothing** (essential to prevent zero probabilities):

$$\theta_{j,k} = \frac{N_{j,k} + \alpha}{N_k + 2\alpha}$$

  where $\alpha > 0$ (typically $\alpha = 1$)
- **Applications:** Document classification (bag-of-words with binary features), medical diagnosis, binary feature sets

## Count Data: Multinomial Naïve Bayes

- **Use case:** Discrete frequency counts (e.g., word counts in documents)
- **Assumption:** Feature vector $\mathbf{X} = (f_1, \dots, f_p)$ follows multinomial distribution
- Likelihood (ignoring constant combinatorial factor):

$$P(\mathbf{X} \mid Y = k) \propto \prod_{j=1}^{p} \phi_{j,k}^{f_j}$$

  where $\phi_{j,k}$ = probability of term $j$ occurring in class $k$
- Normalization constraint: $\sum_{j=1}^{p} \phi_{j,k} = 1$

## Multinomial NB: Parameter Estimation

- Parameter estimation with Laplace smoothing:

$$\phi_{j,k} = \frac{\sum_{i:y_i=k} f_{ij} + \alpha}{\sum_{i:y_i=k} \left(\sum_{m=1}^{p} f_{im}\right) + \alpha p}$$

  - $f_{ij}$ = count of feature $j$ in instance $i$
  - $\alpha > 0$ (smoothing parameter, typically $\alpha = 1$)
  - $p$ = number of features
- Posterior probability:

$$P(Y = k \mid \mathbf{X}) \propto P(Y = k) \prod_{j=1}^{p} \phi_{j,k}^{f_j}$$

- **Key difference from Bernoulli:**
  - Multinomial: Uses frequency counts (multiple occurrences matter)
  - Bernoulli: Uses binary presence/absence (multiple occurrences don't matter)

## Multinomial NB: Applications & Notes

- **Applications:** Text classification, spam detection, categorical data analysis
- **Important implementation details:**
  - Typically uses log probabilities to avoid underflow
  - Ignores non-occurring features ($\prod$ only over present features)
  - Handles varying document lengths through normalization
- **Variant:** Complement Naïve Bayes - improves performance on imbalanced text datasets

## Example on "Default" Data with Naive Bayes

**Dataset:** "Default.csv" contains 10 000 observations of credit card users.

- Binary outcome `default`: "Yes" vs. "No" (encoded as `default_enc` = 1/0)
- Categorical predictor `student`: "Yes" vs. "No" (encoded as `student_enc` = 1/0)
- Continuous predictors `balance` and `income`

**Part 1: Bernoulli Naive Bayes (Student → Default)**

- *Predictor:* `student_enc` $\in \{0, 1\}$
- *Response:* `default_enc` $\in \{0, 1\}$
- *Train/test split:* 70% train, 30% test, random_state=1, shuffle=True
- *Threshold:* $P(\text{default} = 1 \mid \texttt{student\_enc}) > 0.2$
- *Procedure:*
  1. Fit `BernoulliNB` on $\{\texttt{student\_enc}, \texttt{default\_enc}\}_{\text{train}}$.
  2. Compute $P(\text{default} = 1 \mid \texttt{student\_enc} = 0 \text{ or } 1)$.
  3. Classify test point as "default" if its posterior $> 0.2$, otherwise "no default."
  4. Compute confusion matrix and metrics (accuracy, precision, recall, etc).
- *Result (example):*

$$\text{Confusion Matrix} = \begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}, \quad \{\text{accuracy}, \text{precision}, \text{recall}, \text{F}_1\}$$

**Part 2: Gaussian Naive Bayes (Balance + Income → Default)**

- *Predictors:* `balance` and `income` (continuous)
- *Response:* $\texttt{default\_enc} \in \{0, 1\}$
- *Train/test split:* same 70/30 split as above
- *Threshold:* $P(\text{default} = 1 \mid \texttt{balance}, \texttt{income}) > 0.5$
- *Procedure:*
  1. Fit `GaussianNB` on $\{\texttt{balance}, \texttt{income}, \texttt{default\_enc}\}_{\text{train}}$.
  2. Compute posterior $P(\text{default} = 1 \mid \texttt{balance}, \texttt{income})$ for each test point.
  3. Classify as "default" if posterior $> 0.5$, else "no default."
  4. Compute confusion matrix and metrics (accuracy, precision, recall, etc).
- *Result (example):*

$$\text{Confusion Matrix} = \begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}, \quad \{\text{accuracy}, \text{precision}, \text{recall}, \text{F}_1\}$$

## Naïve Bayes: Algorithm Comparison

| **Type** | **Feature Space** | **Likelihood** | **Parameters** |
|---|---|---|---|
| Gaussian | Continuous | Normal | $\mu_{j,k}, \sigma_{j,k}$ |
| Bernoulli | Binary | Bernoulli | $\theta_{j,k}$ |
| Multinomial | Counts | Multinomial | $\phi_{j,k}$ |

- **Shared characteristics:**
  - Conditional independence assumption
  - Requires smoothing for discrete features
  - Fast training ($\mathcal{O}(np)$ time)
  - Optimal when independence assumption holds
- **Weakness:** Feature correlation reduces performance

## Fundamental Differences

| Aspect | Naïve Bayes Classifier (NBC) | K-Nearest Neighbors (KNN) |
|---|---|---|
| **Type** | Probabilistic generative model<br>Makes distributional assumptions | Instance-based model<br>Makes no distributional assumptions |
| **Learning** | Eager learner:<br>Builds model during training | Lazy learner:<br>Defers computation to prediction time |
| **Decision** | Bayesian decision theory: Chooses class with highest posterior probability | Majority vote:<br>Chooses class most frequent among neighbors |
| **Assumptions** | Conditional feature independence given class (naïve assumption) | Local consistency:<br>Similar inputs have similar outputs |

## Advantages & Disadvantages

**Naïve Bayes**

+ Extremely fast prediction
+ Handles high dimensions well
+ Works with small datasets
+ Naturally handles missing data
+ Probabilistic outputs
– Strong independence assumption
– Poor with correlated features
– Limited representation capacity
– Sensitive to irrelevant features

**K-Nearest Neighbors**

+ No training time
+ Simple to implement
+ No parametric assumptions
+ Naturally handles multi-modal classes
– Slow prediction time
– Memory intensive
– Sensitive to irrelevant features
– Needs careful distance metric choice
– Curse of dimensionality

## When to Use Which?

**Choose Naïve Bayes when:**

- Real-time prediction is needed
- Training data is large but resources limited
- Features are approximately independent
- Dataset has high dimensionality
- You need probability estimates
- Text classification problems

**Choose KNN when:**

- Decision boundary is irregular
- You have low-dimensional data
- Dataset size is moderate
- Features are equally important
- You need simple implementation
- Data is noise-free and normalized

**Hybrid approach:** Use NBC for initial fast screening, KNN for uncertain cases

## Practical Performance Comparison

| Characteristic | NBC Speed | KNN Speed | NBC Accuracy | KNN Accuracy |
|---|---|---|---|---|
| Small datasets ($n < 1,000$) | Fast | Fast | Variable | Often high |
| Large datasets ($n > 100,000$) | **Very fast** | **Slow** | Good | **Very good** |
| High dimensions ($p > 1,000$) | **Excellent** | **Poor** | Good | **Poor** |
| Irrelevant features | Sensitive | Sensitive | Degrades | Degrades |
| Correlated features | Problematic | OK | Degrades | Robust |
| Non-linear boundaries | Struggles | Excellent | Low | High |

Table: Performance characteristics: NBC vs KNN

**Note:** NBC often outperforms KNN in text classification, while KNN excels in computer vision (with dimensionality reduction) **Source:** Empirical observations from:

- Zhang, H. (2004). *The optimality of Naive Bayes.* AA1, 1(2), 3.
- Cover, T. M., & Hart, P. E. (1967). *Nearest neighbor pattern classification.* IEEE Transactions on Information Theory, 13(1), 21-27.
- Hand, D. J., & Yu, K. (2001). *Idiot's Bayes—not so stupid after all?* International Statistical Review, 69(3), 385-398.

## Assignment 12

Answer the following questions in Jupyter Notebook format. Show your code and briefly interpret each result. Upload the completed `.ipynb` to K-LMS by next Tuesday at midnight.

**Q1: KNN on** `student-mat.csv`

- Use `G3` as response and {`age`, `Medu`, `Fedu`, `traveltime`, `studytime`, `failures`, `famrel`, `freetime`, `goout`, `Dalc`, `Walc`, `absences`} as predictors.
- Split 70/30 (random_state=6), then for $k \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19\}$: train a `KNeighborsRegressor`, record the accuracy on train and test sets.
- Plot $k$ vs. train/test accuracy; identify the $k$ with best test performance.

**Q2: Naive Bayes on** `Default.csv`

- Encode `default_enc` = (`default` == "Yes") and `student_enc` = (`student` == "Yes").
- *(a) BernoulliNB:* use {`student_enc`} → `default_enc`, split 70/30 (random_state=1), threshold 0.6. Report confusion matrix and {accuracy, precision, recall, F1}.
- *(b) GaussianNB:* use {`balance`, `income`} → `default_enc`, same split, threshold 0.6. Report confusion matrix and metrics.
- Compare which feature set yields better classification, and comment on threshold choice.