Inf620 - Lecture 8 - Naive Bayes

Depto de Informática - UFV

Ricardo Ferreira
ricardo@ufv.br

2025

Introduction

- Class Material (click here for Colab)
- **Review**: Supervised Learning with KNN
- **Problems**: Classification
- Naive Bayes Technique:
  - Variables are independent
  - Simple and fast
  - Probability-based

Review

- **Supervised Learning**
  - Given a set of examples that are [input, output] pairs, an algorithm must find (or learn) a rule that does a good job predicting the output for a new (unseen) input.
- **Classification**
  - Determine which (categorical) class a given observation belongs to.

KNN Review

- **Definition**: Supervised method used for classification and regression.
  - Calculates the distance between the test point and all training points.
  - Selects the k nearest neighbors.
  - Classifies the test point based on the majority of neighbors (for classification) or the average of neighbors (for regression).
- **Common Distance**: Euclidean or Manhattan
- **Choosing the value of k**:
  - Small k: more sensitive to noise.
  - Large k: may overly smooth the decision boundary.
- **Advantages**:
  - Simplicity and intuition.
  - Effective for data with clear class separation.
- **Disadvantages**:
  - High computational cost for large datasets.
  - Sensitive to data scaling.

KNN Challenges in High Dimensions

- **Curse of Dimensionality**:
  - In high dimensions, the space is vast, making point proximity difficult.
  - In low dimensions, nearby points show a clear difference from the average.
  - In high dimensions, nearby points don't differ much from the average, weakening the notion of "closeness".

Example of the Curse of Dimensionality

- **Practical Example**:
  - Consider the distance between people using only a few dimensions, such as weight and height: it's easy to find similar people.
  - Adding more dimensions (age, country of birth, profession, interests, etc.) makes it harder to find very close or very distant people.
  - With more dimensions, everyone becomes, on average, equally distant, and the notion of proximity is lost.

Supplementary Material

- Python Data Science Handbook by Jake VanderPlas (click here)

Naive Bayes: Overview

- **Definition:** Classification method based on probability
- **Key characteristics:**
    - Assigns class labels to instances
    - Uses attribute values for classification
    - Labels are drawn from a finite set

Fundamental Principle of Naive Bayes

- **Independence Assumption:**
  - Each attribute contributes independently to the classification
  - Does not consider correlations between attributes

- **Example:** Classifying an apple
  - Attributes: color (red), shape (round), size (10cm)

  - Each feature is considered independently

Comparison: Naive Bayes vs. KNN

Naive Bayes

- Probability-based
- Assumes independence
- Fast training
- Good for categorical data

KNN

- Distance-based
- Considers spatial relationships
- No training phase
- Good for numerical data

Advantages and Disadvantages

- **Naive Bayes**
  - + Simple and fast
  - + Good for small datasets
  - - Assumes independence (not always true)
  - Despite its limitations, it can be very useful in many practical applications.
- **KNN**
  - + Intuitive
  - + Makes no assumptions about the data
  - - Computationally intensive

Advantages and Disadvantages

- **Naive Bayes**
  - $+$ Great as an initial baseline
  - $+$ Very fast in both training and prediction
  - $+$ Provides direct probabilistic predictions
  - $+$ Often very easy to interpret
  - $+$ Has few (or no) tunable parameters
  - - Assumes independence (not always true)

What is the Naive Bayes Classifier?

- **Naive Bayes** is a classification method based on Bayes' Theorem.
- Assumes all attributes are **independent**, which simplifies the calculation of probabilities.
- Widely used for **text** classification, such as **spam** detection in emails.

Example: Classifying Email as Spam or Normal

- We have **12 emails**, where:
  - 8 are **normal**
  - 4 are **spam**
- Some characteristic words:
  - Normal emails: **dear, friend**
  - Spam emails: **money, free**

Probability of an Email Being Spam

- We want to classify an email that contains the words **money** and **free**.
- We need to calculate:

$$P(\text{spam}|\text{money, free}) \quad \text{and} \quad P(\text{normal}|\text{money, free})$$

- We use Naive Bayes to calculate the probabilities for each class.

Calculating Probabilities

- Prior probability of **spam** and **normal**:

$$P(\text{spam}) = \frac{4}{12} = 0.33 \quad \text{and} \quad P(\text{normal}) = \frac{8}{12} = 0.67$$

- Probability of the words in the email:

$$P(\text{money}|\text{spam}),\ P(\text{free}|\text{spam}),\ P(\text{money}|\text{normal}),\ P(\text{free}|\text{normal})$$

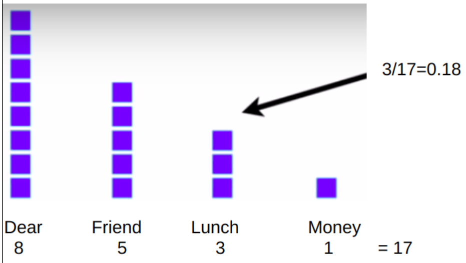
Final Calculation for Classification

$$P(\text{spam}|\text{money, free}) \propto P(\text{money}|\text{spam}) \cdot P(\text{free}|\text{spam}) \cdot P(\text{spam})$$

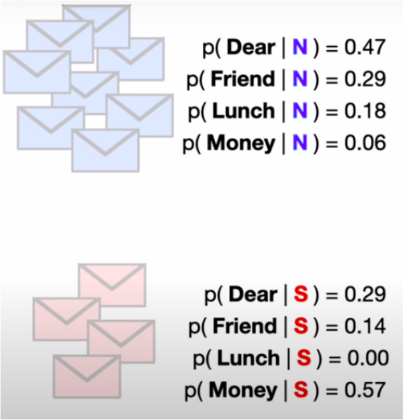
$$P(\text{normal}|\text{money, free}) \propto P(\text{money}|\text{normal}) \cdot P(\text{free}|\text{normal}) \cdot P(\text{normal})$$

- We classify the email as **spam** or **normal** based on the highest probability.

Naive Bayes

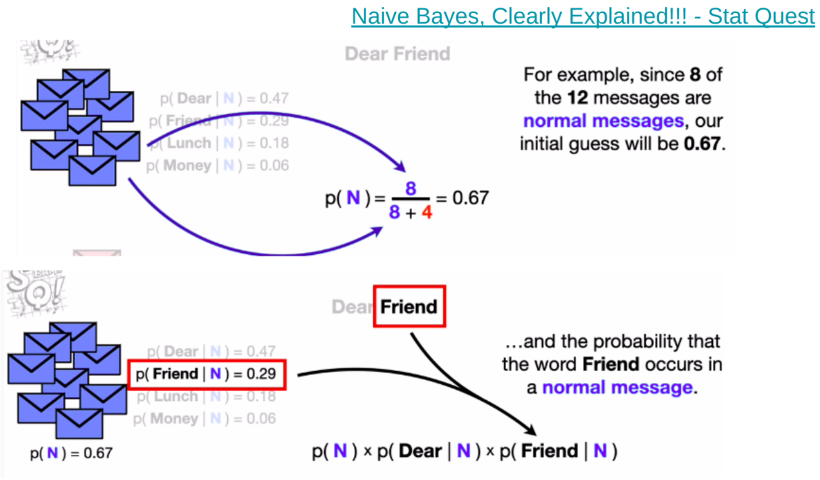


[Naive Bayes, Clearly Explained!!! - Stat Quest]



Clique aqui Stat Quest

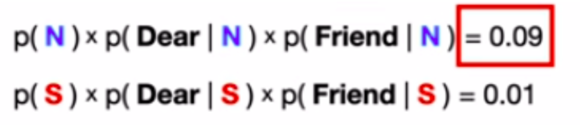
Naive Bayes

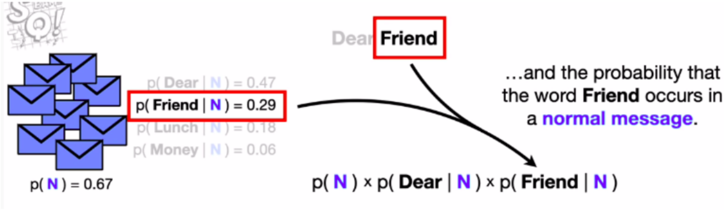


Clique aqui Stat Quest

Naive Bayes

[Naive Bayes, Clearly Explained!!! - Stat Quest]

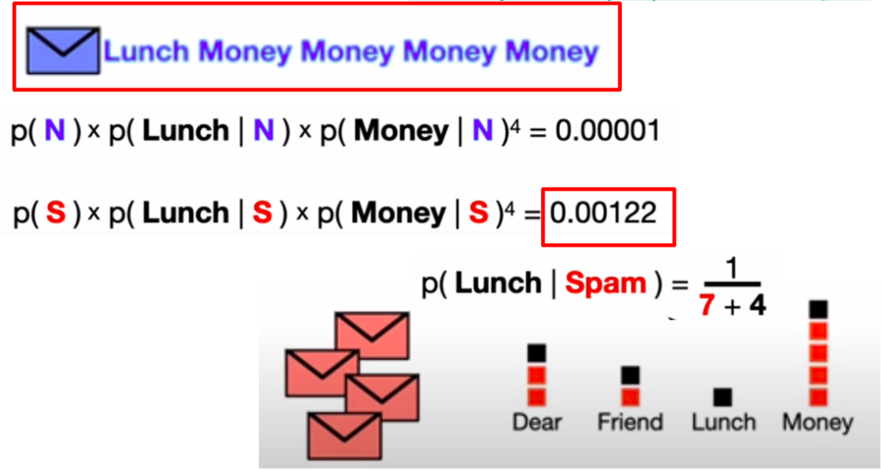




Clique aqui Stat Quest

Naive Bayes

[Naive Bayes, Clearly Explained!!! - Stat Quest]

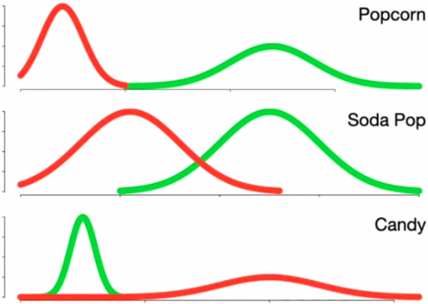


Clique aqui Stat Quest

Probability Distribution

Gaussian Naive Bayes - See Stat Quest....

Popcorn (grams)	Soda Pop (ml)	Candy (grams)
24.3	750.7	0.2
28.2	533.2	50.5
etc.	etc.	etc.



ll 2

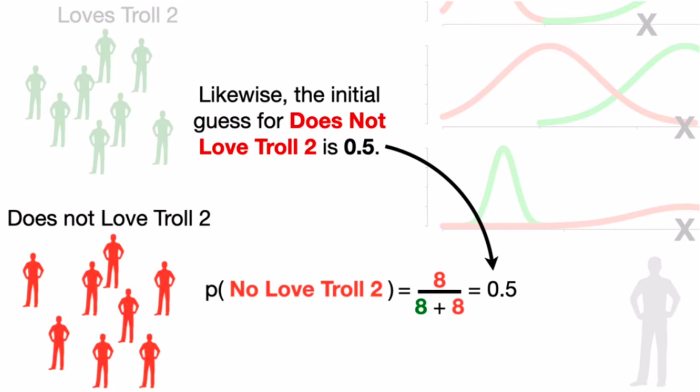
Popcorn (grams)	Soda Pop (ml)	Candy (grams)
2.1	120.5	90.7
4.8	110.9	102.3
etc.	etc.	etc.

Gaussian Naive Bayes is named after the **Gaussian** distributions that represent the data in the **Training Dataset.**

Click here: Stat Quest

Probability Distribution

Gaussian Naive Bayes - See Stat Quest....



Click here: Stat Quest

K-Nearest Neighbors (KNN) Algorithm

```
KNN Algorithm:
1. Receive the training dataset with n examples
2. Define the value of k (number of neighbors)
3. For each test point:
   a. Calculate the distance from the test point to all points
   b. Select the k nearest training points
   c. If it's classification:
      i. Return the most common class among the k neighbors
   d. If it's regression:
      i. Return the average value of the k neighbors
4. End
```

Naive Bayes Algorithm

```
Naive Bayes Algorithm:
1. Receive the training dataset with n examples
2. Calculate the prior probability for each class
3. For each test point:
   a. Calculate the conditional probability of each attribute
      given each class
   b. Multiply the conditional probabilities by the
      prior probability to get the posterior
      probability of each class
   c. Return the class with the highest posterior probability
4. End
```

K-Nearest Neighbors (KNN) Algorithm

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
data = load_iris() # Load dataset
X = data.data
y = data.target
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
knn = KNeighborsClassifier(n_neighbors=3) # Train
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test) # Predict on the test set
accuracy = accuracy_score(y_test, y_pred) # Compute accuracy
print(f'Accuracy: {accuracy:.2f}')
```

Naive Bayes Algorithm

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
data = load_iris() # Load dataset
X = data.data
y = data.target
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
nb = GaussianNB() # Train
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test) # Predict on the test set
accuracy = accuracy_score(y_test, y_pred) # Compute accuracy
print(f'Accuracy: {accuracy:.2f}')
```