

Association Rule Learning & Naïve Bayes

Prepared by Dr. Sumaya Alkubaisi

FNT633
Machine Learning in Finance

Lecture 7



Learning Goals

By the end of this lecture, you will be able to:

- Understand Association Rule Learning and Apriori Algorithm
- Compute Support, Confidence, and Lift
- Apply Apriori in finance datasets using Python
- Understand Naïve Bayes and its application in finance risk prediction
- Apply Naïve Bayes using Python
- Learn when to use Apriori vs Naïve Bayes

Introduction to Association Rule Learning

- Association Rule Learning is a rule-based machine learning technique used to discover relationships between items or events that frequently appear together in large datasets such as transactions (core idea of Market Basket Analysis)
 - It helps answer questions like: “If a customer performs X, how likely are they to perform Y?”
 - It produces rules in the form: IF {Item A} THEN {Item B}
- These rules show how the presence of one item affects the likelihood of another.

Simple Examples:

Retail: IF a customer buys bread, THEN they also buy butter.

Banking: IF a customer has late loan payments, THEN credit score decreases.

Goal: Identify meaningful patterns that help in decision-making (e.g., recommendations, fraud alerts, and understanding buying behavior, etc).

Key Metrics

Association rules are evaluated with 3 main metrics:

Metric	Meaning	Simple Explanation	Example
Support(A→B)	$P(A \cup B)$	How often A and B occur together in all transactions	5% of all transactions include bread & butter together
Confidence(A→B)	$P(B A) = \frac{\text{support}(A \cup B)}{\text{support}(A)}$	If A happens, how likely B also happens?	If A happens, 50% chance B happens.
Lift(A→B)	$\frac{\text{Confidence}(A \rightarrow B)}{\text{support}(B)}$	Measures how much A increases the chance of B occurring	Lift = 1.5 → A & B occur together more than expected → positive association

Example

Total transactions = 100

Support(AUB)=Number of transactions containing both A and B / Total number of transactions

Transactions with A = 20 $\rightarrow$ support(A) = 0.20

Transactions with both A and B = 10 $\rightarrow$ support(A $\cup$ B) = 0.10

Confidence(A $\rightarrow$ B) = 0.10 / 0.20 = 0.5 $\rightarrow$ If A happens, 50% chance B happens.

Lift:

Lift(A $\rightarrow$ B) = Confidence(A $\rightarrow$ B) / support(B) = P(B|A) / P(B).

Lift > 1 $\rightarrow$ A increases the likelihood of B.

Lift Interpretation:

Lift > 1 $\rightarrow$ A and B are positively related

Lift = 1 $\rightarrow$ No relationship

Lift < 1 $\rightarrow$ A reduces likelihood of B (negative relationship)

Association Rules: Practical Examples

Association rules don't replace predictive models, but they enhance analysis by revealing hidden patterns and relationships

- **General**

Rule: If a user creates a new account and changes password within 10 minutes → then logs in from a new country

Interpretation: This combination may indicate account takeover

- **Retail**

Rule: IF {Customer buys bread and milk} → THEN {Customer also buys butter}

Interpretation: Customers who buy bread and milk often also buy butter.

Association Rules: Practical Examples

- **Financial**

Rule: IF {Many small online purchases at night} → THEN {Chargeback next day}

Interpretation: A pattern linked to stolen card testing

- **Banking**

Rule: IF {New device} AND {High transfer amount} → THEN {Failed OTP attempts}

Interpretation: Suspicious behavior possibly related to fraud

- **E-commerce**

Rule: IF {Different shipping & billing address} → THEN {Order cancellation after 24 hours}

Interpretation: Often associated with fraudulent orders

Market Basket Analysis (MBA) Overview

- Technique to find items that are frequently bought together.
- Purpose: Understand customer behavior and detect patterns.

Examples:

Retail: IF {Bread & Milk} → THEN {Butter}

Finance: IF {High online purchase at night & New device} → THEN {Chargeback}

Connection to Apriori: Apriori is the algorithm used to discover these frequent co-occurring items.

- The Apriori algorithm finds frequent item combinations step-by-step, starting from single items and expanding only when combinations are frequent.

Key idea (Apriori property):

- If a set of items is frequent, then all of its smaller subsets must also be frequent.
(So we don't waste time checking itemsets that can't possibly be frequent.)

How it works in simple words:

- Start with 1-item sets and find which items are frequent.
- Use those to build 2-item sets, and keep only the frequent ones.
- Use frequent 2-item sets to build 3-item sets, and repeat.
- Stop when no new frequent itemsets can be found.

Market Basket Analysis Example (Using Apriori Algorithm)

-Suppose we have 5 transactions:

TID	Items Bought
T1	Bread, Milk
T2	Bread, Butter
T3	Milk, Butter
T4	Bread, Milk, Butter
T5	Bread, Milk

-Assume minimum **support** = **40%** (2 out of 5 transactions): must appear at least twice for us to keep it.

Market Basket Analysis Example (Using Apriori Algorithm)

Step 1: Find Frequent 1-Itemsets

- Count each item in all transactions
- Keep items meeting minimum support threshold
- Count each item: {Bread}: 4, {Milk}: 4, {Butter}: 3

All appear ≥ 2 times $\rightarrow$ keep all.

Frequent 1-itemsets: {Bread}, {Milk}, {Butter}

Market Basket Analysis Example (Using Apriori Algorithm)

Step 2: Generate and Check 2-Itemsets

Combine the frequent 1-itemsets:

{Bread, Milk}: 3

{Bread, Butter}: 2

{Milk, Butter}: 2

All meet support → keep all.

Frequent 2-itemsets: {Bread, Milk}, {Bread, Butter}, {Milk, Butter}

Market Basket Analysis Example (Using Apriori Algorithm)

Step 3: Generate and Check 3-Itemsets

Combine frequent 2-itemsets:

{Bread, Milk, Butter}: 2

Support $\geq 2 \rightarrow$ keep.

Frequent 3-itemset: {Bread, Milk, Butter}

Step 4: Stop

No larger itemsets can be formed. Algorithm ends.

Result

Frequent patterns found:

Single items: Bread, Milk, Butter

Pairs: (Bread & Milk), (Bread & Butter), (Milk & Butter)

Triple: (Bread, Milk, Butter)

Support, Confidence, Lift in MBA Context

- **Support:** How often items occur together (Bread & Milk = $3/5 = 60\%$)
- **Confidence:** How often B occurs given A (Bread $\rightarrow$ Milk = $3/4 = 75\%$)
- **Lift:** How much A increases likelihood of B (Lift $> 1 \rightarrow$ positive association)

Chapter
7

Summary

- **Association Rule Learning:** find frequent patterns
- **Apriori Algorithm:** builds larger itemsets iteratively
- **Support, Confidence, Lift:** quantify rule strength
- **Example applications:** retail, finance, anomaly detection

Synthetic Dataset — Finance Transactions

TID	Items Bought
T1	Credit Card Payment, Online Purchase
T2	ATM Withdrawal, Bill Payment
T3	Online Purchase, Mobile Recharge
T4	Credit Card Payment, Online Purchase
T5	Online Purchase, Bill Payment

❖ Notes:

TID = transaction ID

Items Bought = items or services in that transaction

Preprocessed data: binary matrix (one-hot) ready for Apriori, this format is required by `mlxtend.frequent_patterns.apriori`

One-hot encoded: each column represents an item/service, and each cell has 1 (present) or 0 (absent).

Python Demo: Apriori with Finance transaction

**#Goal: perform Market Basket Analysis to find frequent patterns and
association rules**

```
import pandas as pd
```

Create Finance transactions dataset: e.g., multiple product/services purchased in a single #session

```
data = {  
    'TID': ['T1','T2','T3','T4','T5'],  
    'Credit Card Payment': [1,0,0,1,0],  
    'Online Purchase': [1,0,1,1,1],  
    'ATM Withdrawal': [0,1,0,0,0],  
    'Bill Payment': [0,1,0,0,1],  
    'Mobile Recharge': [0,0,1,0,0]  
}
```

```
df = pd.DataFrame(data)  
df = df.drop('TID', axis=1)  
df
```

Python Demo: Apriori with Finance transaction

```
from mlxtend.frequent_patterns import apriori, association_rules
```

```
# Find frequent itemsets with min support 0.4
```

```
frequent_itemsets = apriori(df, min_support=0.4, use_colnames=True)
```

```
# Generate association rules
```

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
```

```
rules
```

[mlxtend.frequent_patterns](#): implements Apriori & association rules

[min_threshold=0.5](#) means:

- We only keep rules whose confidence is at least 0.5 (50%).
- So, `min_threshold=0.5` filters out weak rules that happen less than half the time.

Python Demo: Apriori with Finance transaction

#Output

■ Rule 0

Antecedents: Credit Card Payment “If the customer made a credit card payment”

Consequents: Online Purchase “Then they also made an online purchase”

Support: Happens in 40% of all transactions

Confidence: 80% of the time a credit card payment occurs, there is also an online purchase

Lift: The chance of an online purchase increases by 25% when there’s a credit card payment

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(Credit Card Payment)	(Online Purchase)	0.4	0.8	0.4	1.0	1.25
1	(Online Purchase)	(Credit Card Payment)	0.8	0.4	0.4	0.5	1.25

- Customers who make a credit card payment also make an online purchase most of the time. Lift > 1 means this is more likely than random chance.

■ Rule 1

- Online purchases are often associated with credit card payments, but less strongly than the first rule (confidence = 50%).

Naïve Bayes Concepts

- Naïve Bayes is a probabilistic classifier based on Bayes' Theorem. It predicts the class of a data point by calculating the probability of each class and choosing the class with the highest probability.
- Assumes feature independence
- Works well with categorical and numeric data

Key idea: "Given some evidence (features), what is the most likely class?"

Example:

If an email contains words like "free", "win", "prize", Naïve Bayes calculates the probability of the email being spam vs not spam.

Bayes Theorem

$$P(A|B) = P(B|A) P(A) / P(B)$$

Where:

A = class (e.g., high risk)

B = observed features (e.g., transaction type, amount)

Why is it called “Naïve”?

- Because it assumes that all features are independent of each other, which is rarely true in real life.

Example:

Words “free” and “win” often appear together in spam → not independent

But Naïve Bayes still works very well, even with this wrong assumption.

Why and Where is Naïve Bayes Used?

- It is fast, simple, works great on high-dimensional data, and requires very little training data.

Common applications:

Use Case

Example

Spam detection

Gmail spam filter

Sentiment analysis

Positive / negative reviews

Text classification

Categorizing documents

Medical diagnosis

Predicting diseases from symptoms

- Naïve Bayes is a fast probability-based classifier that works extremely well for text and classification problems, and is different from models like KNN or Decision Trees because it uses probabilities—not distances or rule-splits—to make predictions.

Types of Naïve Bayes

- **Gaussian:** continuous features (e.g., Age, income, temperature)
- **Multinomial:** counts/frequencies (text, categorical) (e.g., Word counts in text, number of clicks)
- **Bernoulli:** binary features (e.g., Yes/No, True/False, Word present or not)

Types of Naïve Bayes

Pros & Cons

Pros: Simple, fast, handles high-dimensional data

Cons: Assumes independence (not always true), may underperform with correlated features

Finance Use Case

Predict loan default or risk category based on customer features

Features: income, age, account history, credit score

Python Demo: Naïve Bayes

Step 1: Synthetic Loan Default Dataset

```
import pandas as pd  
import numpy as np
```

```
# 1. Create Synthetic Dataset
```

```
np.random.seed(42)
```

```
n = 500
```

```
data = pd.DataFrame({  
    "Age": np.random.randint(21, 65, n),  
    "Income": np.random.randint(2000, 15000, n),  
    "LoanAmount": np.random.randint(500, 25000, n),  
    "CreditScore": np.random.randint(300, 900, n),  
    "EmploymentStatus": np.random.choice(["Salaried", "Self-Employed", "Unemployed"], n, p=[0.6, 0.25,  
0.15])  
})
```

Python Demo: Naïve Bayes

Step 1: Synthetic Loan Default Dataset

Label: Default risk (1 = Default, 0 = No Default)

```
data["Default"] = (  
    (data["CreditScore"] < 580) |  
    (data["LoanAmount"] > 15000) |  
    (data["EmploymentStatus"] == "Unemployed")  
)<div data-bbox="775 927 950 960" data-label="Page-Footer">

www.midocean.ae


```

Python Demo: Naïve Bayes

Step 1: Synthetic Loan Default Dataset

2. Save Dataset

```
file_path = "/content/loan_data.csv"  
data.to_csv(file_path, index=False)  
print("Dataset saved to:", file_path)
```

3. Load Dataset -

```
df = pd.read_csv("/content/loan_data.csv")  
df.head()
```

Python Demo: Naïve Bayes

Step 2: Preprocessing the Loan Default Dataset

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report

# Encode categorical feature
le = LabelEncoder()
df["EmploymentStatus"] = le.fit_transform(df["EmploymentStatus"])
```

Python Demo: Naïve Bayes

Step 2: Preprocessing the Loan Default Dataset

```
# Split features & label
```

```
X = df.drop("Default", axis=1)
```

```
y = df["Default"]
```

```
# Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Python Demo: Naïve Bayes

Step 2: Train Gaussian Naïve Bayes

```
# Train Naïve Bayes
```

```
model = GaussianNB()
```

```
model.fit(X_train, y_train)
```

```
# Predict & Evaluate
```

```
y_pred = model.predict(X_test)
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Python Demo: Naïve Bayes

Output

Accuracy: 0.91

91% of all predictions were correct.

This is generally good, but accuracy alone can be misleading if the classes are imbalanced (here, more 1s than 0s).

Precision = Of all predicted as this class, how many were correct.

Class 0: 95% of times it predicted 0, it was correct.

Class 1: 90% of times it predicted 1, it was correct.

Recall = Of all actual instances of this class, how many were correctly identified.

Class 0: Only 71% of actual 0s were correctly predicted → some 0s were misclassified as 1.

Class 1: 99% of actual 1s were correctly predicted → very few 1s were missed.

F1-score = Harmonic mean of precision and recall (balances them).

Class 0: 0.82 → okay but lower than class 1.

Class 1: 0.94 → very strong.

Support = Number of true instances in the test set.

28 zeros, 72 ones → class imbalance is present.

Accuracy: 0.91					
Classification Report:	precision	recall	f1-score	support	
0	0.95	0.71	0.82	28	
1	0.90	0.99	0.94	72	
accuracy			0.91	100	
macro avg	0.93	0.85	0.88	100	
weighted avg	0.91	0.91	0.91	100	

Python Demo: Naïve Bayes

Output

Key Insights

- Model is very good at detecting defaults (1).
- Model struggles a bit with non-defaults (0) (recall is 0.71).
- Accuracy is high, but class imbalance hides that class 0 predictions are weaker.
- If predicting non-defaults correctly is important, you may need resampling, class weights, or tuning.

➤ For Naïve Bayes, the main tuning parameters are `var_smoothing` (GaussianNB) or `alpha` (Multinomial/BernoulliNB).

Most of the time, data preprocessing and feature selection have more impact than hyperparameter tuning.

Accuracy: 0.91					
Classification Report:					
	precision	recall	f1-score	support	
0	0.95	0.71	0.82	28	
1	0.90	0.99	0.94	72	
accuracy			0.91	100	
macro avg	0.93	0.85	0.88	100	
weighted avg	0.91	0.91	0.91	100	

Python Demo: Naïve Bayes

code explaining

np.random.choice: This is a NumPy function used to randomly select elements from a list or array. Its general form is:

```
np.random.choice(a, size=None, replace=True, p=None)
```

a: The array or list to choose from (e.g., ["Salaried", "Self-Employed", "Unemployed"]).

size: How many elements you want to pick (here n = 500).

replace: Whether an element can be picked more than once (default True).

p: Optional probabilities for each element (must sum to 1).

❖ np.random.choice creates random categorical values, and n controls the number of samples, while p controls the probability of each category appearing.

Practical Guidance

- **Use Apriori:** when interested in frequent patterns, co-occurrences
- **Use Naïve Bayes:** when predicting a target variable from features

Combined Example in Finance

Fraud detection workflow:

- Apriori: identify suspicious transaction patterns
- Naïve Bayes: predict high-risk accounts

Recap

- Association Rule Learning → find patterns & frequent itemsets
- Apriori Algorithm → iterative approach, use support/confidence/lift
- Naïve Bayes → probabilistic classifier, assumes feature independence
- Python makes both easy to implement and visualize
- Choose Apriori for patterns, Naïve Bayes for predictions

References

- Jannes Klaas — Machine Learning for Finance
- Marcos López de Prado — Advances in Financial Machine Learning
- IBM Coursera — Machine Learning with Python for Finance
- mlxtend documentation — <https://rasbt.github.io/mlxtend/>
- scikit-learn docs