From Static to Dynamic: Evolving Bayesian Network Thinking for Real-World Applications

Applied Bayesian Networks: Bridging Theory, Modeling, and Forecasting in Practice

Shenggang Li 14 min read · 1 day ago

Photo by Abi Ghouta Timur on Unsplash

Imagine you're a supply-chain manager trying to predict equipment failures before production halts. Begin by mapping key factors — machine age, maintenance history, and operating temperature — into a static Bayesian network. This snapshot helps quickly estimate breakdown risks based on current data without advanced statistics.

To forecast evolving risks as conditions change, dynamic Bayesian networks extend your static model across multiple time steps. This allows you to anticipate how today's conditions impact future breakdown risks, providing actionable forecasts.

This guide covers both approaches. You'll learn how static networks leverage your knowledge and historical data for immediate, clear risk assessments in fields like credit scoring or fault diagnosis. Then you'll see how dynamic networks handle scenarios like demand forecasting or patient monitoring, highlighting when each method is most effective.

By the end, you'll understand key concepts such as conditional independence and time-slice factorization, and you'll confidently build, test, and use *Bayesian* networks with clear steps and practical code — without complicated theory.

How Static Bayesian Networks Work

Use Case

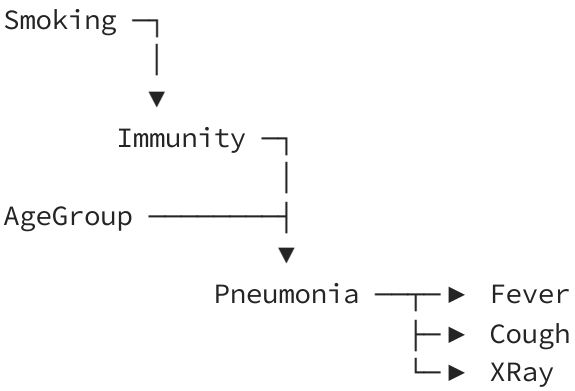
Imagine a hospital triage team that must decide, the moment a patient arrives, whether they likely have community-acquired pneumonia. A static Bayesian network (*BN*) helps by turning each clinical variable — age, smoking history, fever, cough severity and X-ray result — into a node in a directed acyclic graph.

Each node carries a small conditional probability table (*CPT*) that quantifies "given these parent factors, what is the chance of X"?

For example:

Node (discrete states)	Why include it?
Smoking ∈ {Yes, No}	Chronic smokers have poorer lung defences.
AgeGroup ∈ {Child, Adult, Senior}	Seniors and very young have higher risk.
Immunity ∈ {Good, Weak}	Captures diabetes, chemo, HIV, etc.
Pneumonia ∈ {Yes, No}	Hidden condition we want to infer.
Fever ∈ {High, Low, None}	Symptom influenced by infection.
Cough ∈ {Severe, Mild, None}	Another symptom.
X-ray ∈ {Abnormal, Normal}	Imaging evidence (may be delayed).

Causal Plot (directed-acyclic graph):



Risk drivers — AgeGroup and Smoking feed into Immunity; all three feed the latent disease node *Pneumonia*.

Evidence — If *Pneumonia* = *Yes*, the probabilities of high fever, severe cough, and abnormal X-ray all jump.

This graph reflects doctor's knowledge: smoking history, age, and immune status jointly influence the risk of pneumonia; once pneumonia is present, it generates measurable symptoms — fever and cough — which in turn affect chest X-ray findings. By the factorization theorem, because each node is

conditionally independent of its non-descendants given its parents (the local Markov property), the full joint distribution factorizes as:

$$P(S, A, I, P, F, C, X) = P(S)\,P(A)\,P(I)\,P(P \mid S, A, I)\,P(F \mid P)\,P(C \mid P)\,P(X \mid F, C)$$

Think a large spreadsheet listing every combination of seven variables — Smoking, AgeGroup, Immunity, Pneumonia, Fever, Cough, XRay — and storing a probability for each. That quickly grows into hundreds of rows.

We now "divide and conquer" with small lookup tables called *CPTs*. For instance, *P(Smoking)* needs just two numbers, *P(AgeGroup)* just three, and *P(Pneumonia | Smoking, AgeGroup, Immunity)* only one probability per combination of those three (*2 × 3 × 2* entries). Adding the *CPTs* for Fever, Cough, and XRay typically brings us to a few dozen entries total — an order of magnitude smaller than the full joint table.

This approach reduces the number of parameters you must estimate, it still respects the rule that, say, Fever depends only on Pneumonia — not on Smoking or AgeGroup directly — just as the *DAG* specifies.

From medical records — electronic health records (*EHR*) or clinical trial databases — we have a sample of patient histories to estimate these *CPTs* via maximum-likelihood. Suppose we pull many de-identified cases:

Smoking	AgeGroup	Immunity	Pneumonia	Fever	Cough	XRay
No	Adult	Good	No	Low	Mild	Normal
Yes	Senior	Weak	Yes	High	Severe	Abnormal
No	Child	Good	No	None	None	Normal
Yes	Adult	Weak	Yes	High	Mild	Abnormal
No	Senior	Good	No	Low	Mild	Normal

We now compute frequencies. For example, if *40%* of patients are smokers, *35%* are adults with weakened immunity, and so on, the *CPTs* might look like:

Smoking	P(Smoking)
No	0.60
Yes	0.40

AgeGroup	P(AgeGroup)
Child	0.25
Adult	0.50
Senior	0.25

Immunity	P(Immunity)
Good	0.70
Weak	0.30

Pneumonia → Fever	P(Fever=High)	P(Fever=Low)	P(Fever=None)
Yes	0.80	0.15	0.05
No	0.10	0.20	0.70

Pneumonia → Cough	P(Cough=Severe)	P(Cough=Mild)	P(Cough=None)
Yes	0.50	0.30	0.20
No	0.05	0.15	0.80

Fever, Cough → XRay	P(XRay=Abnormal)
(High, Severe)	0.95
(None, None)	0.02

How the BNs is useful in practice

Before labs/X-ray: Enter quick observations — “Adult, Smoker, Weak Immunity, Low Fever, Mild Cough”. The *BN* updates:

$$P(\text{Pneumonia} = \text{Yes} \mid \text{evidence}) = 18\%$$

giving triage nurse tags the patient for priority chest X-ray.

After X-ray: New evidence “XRay = Abnormal” is clamped; the posterior jumps to *92%*. The physician starts antibiotics while waiting for culture results.

Sensitivity analysis: Ask “What if the same patient were a non-smoker”? → probability drops to *12%*. That quantifies smoking’s marginal impact and

helps patient counselling.

Resource planning: Aggregating *BNs* outputs over all admissions gives expected daily pneumonia cases — useful for staffing, bed allocation, or budgeting *CT* scans.

Code Test: Static Bayesian network

By storing tables above in a version-controlled CSV or Parquet store, we can reload them at runtime into a production *BNs* engine. Here is how a concise Python experiment might look, using `pgmpy` to both define the network and perform inference:

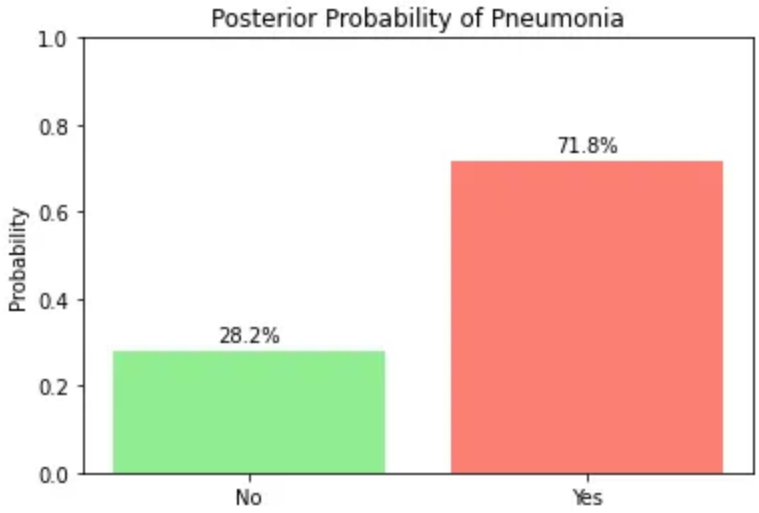
```
import os
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination


raw_df = pd.read_csv('patients_dec.csv', keep_default_na=False)

# Compute CPTs from raw data ---------------------------------------------
# Smoking CPD
sm_counts = raw_df['Smoking'].value_counts(normalize=True).reindex(['No','Yes'])
cpd_smoking = TabularCPD('Smoking', 2,
                         [[sm_counts['No']], [sm_counts['Yes']]],
                         state_names={'Smoking':['No','Yes']})

# AgeGroup CPD
ag_counts = raw_df['AgeGroup'].value_counts(normalize=True).reindex(['Child','Ad
cpd_age = TabularCPD('AgeGroup', 3,
                     [[ag_counts['Child']], [ag_counts['Adult']], [ag_counts['Se
                     state_names={'AgeGroup':['Child','Adult','Senior']})

# Immunity CPD
```

```
im_counts = raw_df['Immunity'].value_counts(normalize=True).reindex(['Good','Wea
cpd_immunity = TabularCPD('Immunity', 2,
                          [[im_counts['Good']], [im_counts['Weak']]],
                          state_names={'Immunity':['Good','Weak']})

# Pneumonia CPD
p_counts = raw_df.groupby(['Smoking','AgeGroup','Immunity','Pneumonia']).size().
p_probs  = p_counts.div(p_counts.sum(axis=1), axis=0).reindex(columns=['No','Yes
values_p = [p_probs['No'].values, p_probs['Yes'].values]
cpd_pneumonia = TabularCPD('Pneumonia', 2, values_p,
                           evidence=['Smoking','AgeGroup','Immunity'],
                           evidence_card=[2,3,2],
                           state_names={
                               'Pneumonia':['No','Yes'],
                               'Smoking':['No','Yes'],
                               'AgeGroup':['Child','Adult','Senior'],
                               'Immunity':['Good','Weak']
                           })

# Fever CPD
f_counts = raw_df.groupby(['Pneumonia','Fever']).size().unstack(fill_value=0)
f_probs  = f_counts.div(f_counts.sum(axis=1), axis=0)
values_f = [f_probs['High'].values, f_probs['Low'].values, f_probs['None'].value
cpd_fever = TabularCPD('Fever', 3, values_f,
                       evidence=['Pneumonia'], evidence_card=[2],
                       state_names={'Fever':['High','Low','None'],
                                    'Pneumonia':['No','Yes']})

# Cough CPD
c_counts = raw_df.groupby(['Pneumonia','Cough']).size().unstack(fill_value=0)
c_probs  = c_counts.div(c_counts.sum(axis=1), axis=0)
values_c = [c_probs['Severe'].values, c_probs['Mild'].values, c_probs['None'].va
cpd_cough = TabularCPD('Cough', 3, values_c,
                       evidence=['Pneumonia'], evidence_card=[2],
                       state_names={'Cough':['Severe','Mild','None'],
                                    'Pneumonia':['No','Yes']})

# XRay CPD
x_counts = raw_df.groupby(['Fever','Cough','XRay']).size().unstack(fill_value=0)
x_probs  = x_counts.div(x_counts.sum(axis=1), axis=0)
values_x = [x_probs['Abnormal'].values, x_probs['Normal'].values]
cpd_xray = TabularCPD('XRay', 2, values_x,
                      evidence=['Fever','Cough'], evidence_card=[3,3],
                      state_names={'XRay':['Abnormal','Normal'],
                                   'Fever':['High','Low','None'],
                                   'Cough':['Severe','Mild','None']})

# Build model & add CPDs ----------------------------------------------
model = BayesianNetwork([
    ('Smoking','Pneumonia'), ('AgeGroup','Pneumonia'), ('Immunity','Pneumonia'),
```

```
    ('Pneumonia','Fever'), ('Pneumonia','Cough'), ('Fever','XRay'), ('Cough','XR
])
model.add_cpds(cpd_smoking, cpd_age, cpd_immunity,
               cpd_pneumonia, cpd_fever, cpd_cough, cpd_xray)
model.check_model()

# Inference ------------------------------------------------------------
infer = VariableElimination(model)
posterior = infer.query(['Pneumonia'],
                        evidence={'Smoking':'Yes','AgeGroup':'Senior',
                                  'Immunity':'Weak','Fever':'High',
                                  'Cough':'Mild','XRay':'Abnormal'})
labels = posterior.state_names['Pneumonia']
probs  = posterior.values

# Print detailed results -----------------------------------------------
print("=== Pneumonia Risk Assessment ===")
for lbl, pr in zip(labels, probs):
    print(f"P(Pneumonia={lbl} | evidence) = {pr:.3f}")

# Plot BN structure ----------------------------------------------------
plt.figure(figsize=(8,6))
pos = nx.spring_layout(model)
nx.draw_networkx_nodes(model, pos, node_color='skyblue', node_size=2000)
nx.draw_networkx_labels(model, pos, font_size=10)
nx.draw_networkx_edges(model, pos, arrows=False)
plt.title("Bayesian Network for Pneumonia Diagnosis")
plt.axis('off')
plt.show()

# Plot posterior as bar chart ------------------------------------------
plt.figure(figsize=(6,4))
bars = plt.bar(labels, probs, color=['lightgreen','salmon'])
plt.ylabel("Probability")
plt.title("Posterior Probability of Pneumonia")
plt.ylim(0,1)
for bar, pr in zip(bars, probs):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
             f"{pr:.1%}", ha='center')
plt.show()

# Friendly explanation -------------------------------------------------
print(f"\nKey takeaway: with these symptoms, there's a {probs[labels.index('Yes'
```

Here are the results:

```
=== Pneumonia Risk Assessment ===
P(Pneumonia=No | evidence) = 0.282
P(Pneumonia=Yes | evidence) = 0.718

Key takeaway: with these symptoms, there's a 71.8% chance of pneumonia.
```



Evaluation

1. *71.8%* **chance of pneumonia**

- Start empirical antibiotics immediately (time-to-antibiotics strongly tied to outcome).
- Order blood cultures & *CBC*; no need to wait for further imaging.
- Admit to ward vs. discharge from *ED*.

2. *28.2%* **chance of "no pneumonia**

- need communicate uncertainty; schedule follow-up X-ray in 24–48 h to confirm clearing.

- If comorbidities (*COPD*) exist, evaluate for alternative diagnoses (e.g., *CHF*).

How Static BNs Empower Decision Making

The code test above show that the static Bayesian network merges weak signals — age, smoking status, mild cough — with strong ones — high fever, abnormal chest film — into a single, calibrated probability of "pneumonia right now".

- **Transparency:** Each conditional-probability table (*CPT*) reveals how every finding affects the odds; e.g., an abnormal film multiplies the odds by ~19×, so clinicians see exactly why the posterior spikes.
- **Speed:** Probabilities update instantly when new data (such as a *CRP* level) arrive; no model refit is needed.
- **Resource planning:** Summing posteriors for all arrivals gives an expected pneumonia head-count, informing bed assignments, isolation rooms, and antibiotic inventory.

How Dynamic Bayesian Network Works?

I will show how a static *Bayesian* network lets clinicians plug today's symptoms, labs and risk factors into a compact model and instantly get a calibrated probability of pneumonia. That "snapshot" score requires no retraining — just enter the data and make your treatment decision on the spot.

A Dynamic *Bayesian* Network adds a time dimension to the same idea. You "unroll" the graph across discrete time steps, linking each variable at time *t* back to its state (and its parents) at time *t−1*.

For example, in finance you might track a stock's closing price, trading volume and interest-rate announcements over a trading week. As you feed in each day's data — yesterday's close, today's volume spike, tomorrow's policy decision — the *DBN* propagates your probability of a price rise forward without ever rebuilding the model. That live, trend-aware forecast lets you adjust orders, rebalance portfolios or tighten stop-losses in real time.

Think of a DBN as a series of flow-charts laid end to end:

- **Boxes** represent the variables we're tracking.
- **Arrows** show which variables directly influence others from one time step to the next.
- **Inside each box** is the probability distribution for that variable at that moment.

Example with two stocks:

```
TSLA-today ——► NVDA-today
```

The arrow means the chance that *NVDA is up today* depends on whether *TSLA is up today*.

The numbers — "if *TSLA* is up, *NVDA* is up with *55%* probability; if *TSLA* is down, the probability falls to *30%*" — are called a conditional-probability table (*CPT*).

DBN Mechanism

A Dynamic *Bayesian* Network (*DBN*) is a *Bayesian* network "unrolled" over discrete time steps *t=0,1,…,T*.

Denote by *X(t) = {X1(t),…,Xn(t)}* the set of variables at time *t*. We build two kinds of directed arcs:

1. **Intra-slice** arcs, matching the static *BNs* structure within each *X^(t)*.
2. **Inter-slice** arcs, from parents in *X^(t−1)* to children in *X^(t)*.

The full joint distribution over *0≤t≤T* then factorizes as

$$P\big(\mathbf{X}^{(0:T)}\big) = P\big(\mathbf{X}^{(0)}\big) \prod_{t=1}^{T} P\big(\mathbf{X}^{(t)} \mid \mathbf{X}^{(t-1)}\big)$$

and each conditional *P(X^(t) | X^(t−1))* further splits into node-wise *CPDs*:

$$P\big(\mathbf{X}^{(t)} \mid \mathbf{X}^{(t-1)}\big) = \prod_{i=1}^{n} P\big(X_i^{(t)} \mid \mathrm{Pa}_{\mathrm{intra}}(X_i^{(t)}), \mathrm{Pa}_{\mathrm{inter}}(X_i^{(t)})\big)$$

For estimation, we use:

- The initial *CPD P(X^(0))* via static *BN* learning on time-zero data.
- The transition *CPDs:*

$$P(X_i^{(t)} \mid \mathbf{X}^{(t-1)})$$

Then we pool pairs:

$$(\mathbf{X}^{(t-1)}, X_i^{(t)})$$

across time series (maximum likelihood or Bayesian estimators).

Inference — computing:

$$P(\mathbf{X}^{(t)} \mid \text{evidence}_{0:t})$$

This can be done by forward filtering:

$$\alpha_t(x_t) = \sum_{x_{t-1}} \alpha_{t-1}(x_{t-1})\, P(x_t \mid x_{t-1})\, P(e_t \mid x_t)$$

where,

$$\alpha_t(x_t) = P(x_t, e_{1:t})$$

We can apply backward smoothing on the unrolled network for posterior query.

Here is an algorithmic form (filtering loop):

```
Initialize α₀(x) = P(x, e₀)
For t = 1…T:
  For each state x_t:
    α_t(x_t) = P(e_t | x_t) * ∑x t-1 [ α t-1(x t-1) * P(x_t | x t-1) ]
Normalize α_t
```

This approach scales linearly in *T* (times the cost of summing over states) and instantly updates your forecast as new observations arrive. This captures the temporal dependencies that static *BNs* cannot.

Where does the word Dynamic come in?

A *DBN* glues *yesterday* and *today* copies of that flow-chart together:

```
          •── self-loop ▶ TSLA-tomorrow
         /
TSLA-today
         \
          •──▶ NVDA-tomorrow
```

Self-loops capture momentum (“up days tend to be followed by up days”). Cross-arrows like *TSLA*-today → *NVDA*-tomorrow capture lead–lag spill-over.

So a *DBN* is a probability machine that tells you:

> *Given everything I saw today, here's the probability each stock will be up tomorrow.*

How to build a DBN?

1. **Data** — Take a few hundred days of prices, turn each day's return into 0 = down, 1 = flat, 2 = up.
2. **Learn the arrows** — A laptop runs a little greedy search ("*hill-climb* + *BIC*") testing which arrows make the historical data most probable without making the chart too messy.
3. **Fill in the numbers** — Count how often each arrow-combination happened (with a bit of smoothing so no probability is exactly zero).
4. **Ready to ask questions.**

Total compute time for *4–5* stocks: seconds.

```
"""
Dynamic Bayesian Network (DBN) demo - English friendly output
Stocks: TSLA · NVDA · AMD · TSM   (last 800 trading days)

What you'll get
---------------
1.  "Momentum" question - If TSLA & NVDA rally today, how likely is AMD to rally
2.  "Shock table"       - Which single-day rally helps TSM the most for tomorrow
3.  "Who-leads-whom"     - Arrow diagram of intra-day causal flow.
4.  Basket probability   - Odds that **all four** finish in their top-tercile to

Libraries: pandas · numpy · yfinance · pgmpy ≥ 0.1.7 · networkx · matplotlib
"""

import warnings, numpy as np, pandas as pd, yfinance as yf
import matplotlib.pyplot as plt, networkx as nx
```

```
warnings.filterwarnings("ignore")

# pgmpy - handle API changes gracefully
from pgmpy.models import DynamicBayesianNetwork as DBN
from pgmpy.estimators import HillClimbSearch, MaximumLikelihoodEstimator
from pgmpy.inference  import DBNInference

# BicScore changed case in old vs new releases
try:
    from pgmpy.estimators import BicScore as _BIC
except ImportError:        # very old pgmpy
    from pgmpy.estimators import BicScore as _BIC

# ------------------------------------------------- #
# Download adjusted closes robustly        #
# ------------------------------------------------- #
def fetch_adj_close(tickers, lookback_days=850):
    df = yf.download(tickers, period=f"{lookback_days}d", progress=False)
    if isinstance(df.columns, pd.MultiIndex):
        df = df.swaplevel(axis=1)  # ticker first
        out = pd.concat([(df[t]["Adj Close"]
                          if "Adj Close" in df[t]
                          else df[t]["Close"])
                         for t in tickers], axis=1)
    else:
        out = pd.concat([(df[f"{t} Adj Close"]
                          if f"{t} Adj Close" in df.columns
                          else df[f"{t} Close"])
                         for t in tickers], axis=1)
    out.columns = tickers
    return out

# ------------------------------------------------- #
# Data                                              #
# ------------------------------------------------- #
TIKS  = ["TSLA", "NVDA", "AMD", "TSM"]
prices = fetch_adj_close(TIKS).dropna().tail(800)
rets   = prices.pct_change().dropna()

# discretise each column into terciles 0/1/2
def tercile(series):
    q = series.rank(pct=True)
    return pd.cut(q, bins=[-0.01, 1/3, 2/3, 1.01],
                  labels=[0, 1, 2]).astype(int)

disc_cols = {c: tercile(rets[c]) for c in rets.columns}
disc      = pd.DataFrame(disc_cols)

# add time level
disc.columns = pd.MultiIndex.from_product([disc.columns, [0]],
```

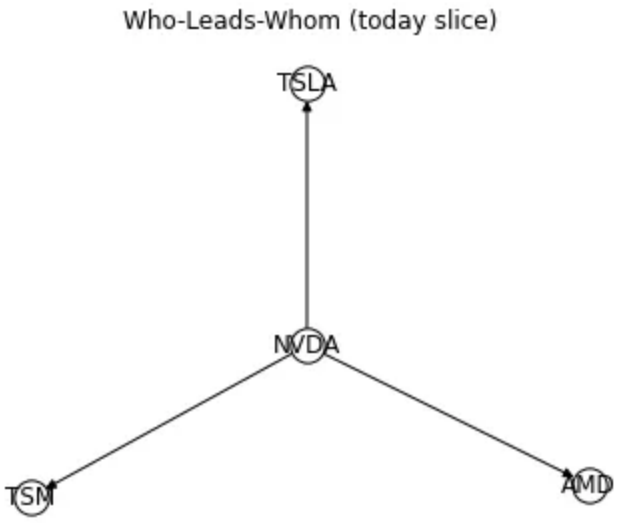
```
                                            names=["sym", "time"])
disc.reset_index(drop=True, inplace=True)

# tomorrow slice (time=1)
lags = disc.copy()
lags.columns = lags.columns.set_levels([1], level="time")
lags.index   = lags.index - 1
disc_t12     = pd.concat([disc, lags], axis=1).dropna().astype(int)

# ------------------------------------------------- #
# Structure learning (handles old/new API styles)    #
# ------------------------------------------------- #
slice0 = disc_t12.xs(0, level="time", axis=1)
score  = _BIC(slice0)
try:                                # old API
    hc      = HillClimbSearch(slice0, scoring_method=score)
    best_bn = hc.estimate()
except TypeError:                   # new API
    hc      = HillClimbSearch(slice0)
    best_bn = hc.estimate(scoring_method=score)

# ------------------------------------------------- #
# Build Dynamic BN                                  #
# ------------------------------------------------- #
dbn = DBN()
for u, v in best_bn.edges():
    dbn.add_edge((u, 0), (v, 0))      # intra-slice
for s in TIKS:
    dbn.add_edge((s, 0), (s, 1))      # self-loops
for u, v in best_bn.edges():          # copy edges forward
    dbn.add_edge((u, 0), (v, 1))

# ------------------------------------------------- #
#  Fit CPTs & make inference object                 #
# ------------------------------------------------- #
dbn.fit(disc_t12)
inf = DBNInference(dbn)

def prob_up(sym, evidence_today):
    """Return [P(down), P(flat), P(up)] for sym at t+1."""
    return inf.query([(sym, 1)], evidence=evidence_today)[(sym, 1)].values

# ------------------------------------------------- #
# Q-1  AMD ↑ given TSLA ↑ & NVDA ↑ today            #
# ------------------------------------------------- #
today = {("TSLA", 0): 2, ("NVDA", 0): 2}
print("\nQ1  P(AMD up) = {:.1f}%".format(prob_up("AMD", today)[2]*100))

# ------------------------------------------------- #
# Q-2  One-day “shock” ranking for TSM              #
```

```
# ---------------------------------------------------- #
baseline = {(s, 0): 1 for s in TIKS}
base_p   = prob_up("TSM", baseline)[2]
deltas = []
for s in TIKS:
    shocked = baseline.copy(); shocked[(s, 0)] = 2
    deltas.append((s, prob_up("TSM", shocked)[2] - base_p))

print("\nQ2  ΔP(TSM up) from single-stock up-shocks:")
for s, d in sorted(deltas, key=lambda x: x[1], reverse=True):
    print(f"   {s}: {d:+.3f}")

# ---------------------------------------------------- #
# Q-3  Draw causal map (“who leads whom”, t = 0)     #
# ---------------------------------------------------- #
# build an nx.DiGraph from the DBN's edge list
G_full = nx.DiGraph()
G_full.add_edges_from(dbn.edges())

# keep only intra-slice edges (time = 0 → time = 0)
edges_today = [(u[0], v[0]) for u, v in G_full.edges() if u[1] == v[1] == 0]

plt.figure(figsize=(5, 4))
nx.draw_networkx(nx.DiGraph(edges_today), arrows=True,
                 node_color="white", edgecolors="black")
plt.title("Who-Leads-Whom (today slice)")
plt.axis("off")
plt.tight_layout()
plt.show()

# ---------------------------------------------------- #
# Q-4  Portfolio ↑ probability tomorrow             #
# ---------------------------------------------------- #
today_ex = {("TSLA",0):0, ("NVDA",0):1, ("AMD",0):1, ("TSM",0):2}
p_port   = np.prod([prob_up(sym, today_ex)[2] for sym in TIKS])
print("\nQ4  P(portfolio up) = {:.1f}%".format(p_port*100))
```

Here are the results:

```
Q1  P(AMD up) = 33.7%

Q2  ΔP(TSM up) from single-stock up-shocks:
```

```
    TSM: +0.021
    AMD: +0.008
    NVDA: +0.005
    TSLA: +0.000

Q4  P(portfolio up) = 1.2%
```



Q1 — P(AMD up) = *33.7%*:

If *TSLA* and *NVDA* both finish in their top-tercile today (“strong up-day”), there’s roughly a 1-in-3 chance *AMD* will also land in its top tercile tomorrow. This is solid — but not overwhelming — follow-through. A momentum-chaser might view this as worth a small tactical bet, not a conviction trade.

Q2 — one-day shock table:

How much does forcing one stock to rally today lift *TSM*’s odds of rallying tomorrow?
• *TSM* itself *+2.1* pts → biggest driver (self-momentum).
• *AMD* +0.8 pts → modest tail-wind.

• *NVDA* +0.5 pts → small positive nudge.
• *TSLA* ≈ 0 pts → essentially no spill-over.
We should focus on *TSM*'s own price action first; keep an eye on *AMD/NVDA* for secondary cues. *TSLA*'s move can be safely ignored when gauging next-day *TSM* strength.

Causal map (plot):

The arrows show the statistical lead-lag pattern the *DBN* discovered within the same day: *NVDA* sits at the center, pointing to *TSLA, AMD*, and *TSM*. *NVDA* often moves before the other three.
If you need an early warning or "leader" to watch intraday, *NVDA* is the best candidate in this quartet.

Q4 — P(all four up) = *1.2%*:
Under today's mixed setup (*TSLA* ↓, *NVDA* ↔, *AMD* ↔, *TSM* ↑) there's only a 1-in-*80* chance that all four will be in their top tercile tomorrow.
The hurdle is very high: we require every name to hit the top third of its distribution. That almost never happens — even four independent *33%* events would give only *1.2%*. This low figure is therefore expected, not alarming.

Key take-aways for a decision-maker

Watch *NVDA* first.
It's the hub in today's causal map; short-term moves in *NVDA* tend to propagate to the other three.

Self-momentum matters most.
TSM rallying today is still the best single predictor of *TSM* rallying tomorrow (+*2.1* pts lift).

***AMD* offers a secondary clue.**
A pop in *AMD* nudges *TSM* up-probability, consistent with supply-chain links.

***TSLA* is largely orthogonal here.**
Its tech/autos story doesn't spill over to foundry/semis in this one-day horizon.

Basket odds look tiny by construction.
The model asks for a *perfect* 4-for-4 top-tercile outcome. Use this as a directional gauge, not a *P&L* forecast.

Takeaway

Static *Bayesian* Networks (*BNs*) simplify combining varied data — such as genetic markers or economic indicators — into clear probability estimates. Rather than multiple complex analyses, *BNs* offer quick, easy-to-interpret snapshots ideal for stable relationships.

Dynamic *Bayesian* Networks (*DBNs*) extend this by modeling how relationships evolve over time, making them effective for forecasting trends like disease progression or market fluctuations.

Real-world applications of static and dynamic *BNs* are broad and practical. In genomics, scientists use them to predict gene responses under different conditions. Economists track how today's trade wars might affect tomorrow's exports. Supply-chain managers forecast sales by factoring in inventory and seasonal trends. Healthcare providers assess drug interaction risks using

patient histories. And marketers create detailed customer profiles by analyzing behaviors over time.

All source code and data are available on GitHub: https://github.com/datalev001/BN_DBN

About me

With over 20 years of experience in software and database management and 25 years teaching IT, math, and statistics, I am a Data Scientist with extensive expertise across multiple industries.

You can connect with me at:

Email: datalev@gmail.com | LinkedIn | X/Twitter

Bayesian | Networking | Machine Learning | Forecasting | Python

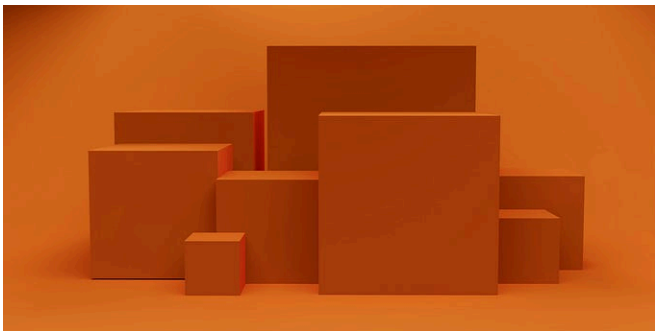
2.6K Followers · 77 Following

No responses yet

Alex Mylnikov

What are your thoughts?

More from Shenggang Li and Towards AI

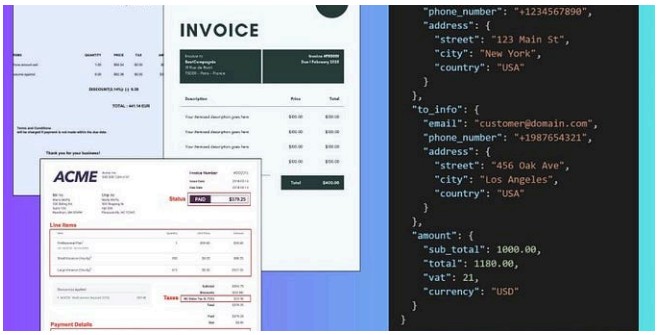


In Data Science Collective by Shenggang Li

Comparing Binary Forecasting Methods for Financial Time...

Introduction

Apr 23 168 1

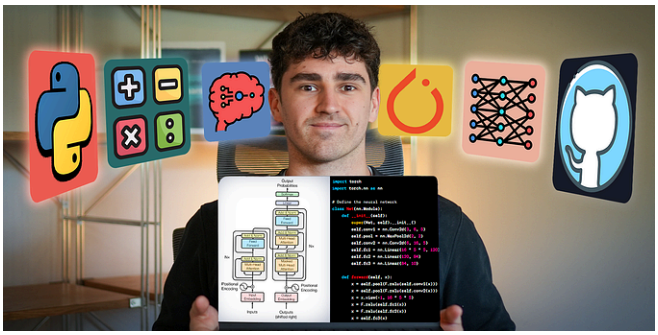


In Towards AI by Jeremy Arancio

Deploy an in-house Vision Language Model to parse millions...

How to build a Document Parsing Pipeline to process millions of documents using Qwen-...

Apr 23 1K 16



In Towards AI by Boris Meinardus

How I'd learn ML in 2025 (if I could start over)

All you need to learn ML in 2025 is a laptop and a list of the steps you must take.

Jan 2 1.93K 53

In Data Science Collective by Shenggang Li

Residual-Tail Boosting: A Fast Post-Training Add-On for Extra...

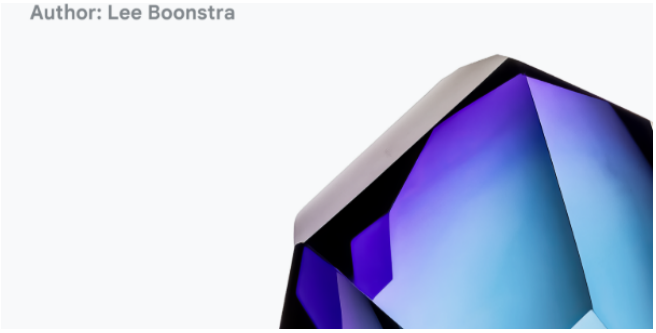
A Practical Error-Correction Method Using Adaptive Residual Deltas with Minimal...

3d ago 190 3

See all from Shenggang Li

See all from Towards AI

Recommended from Medium

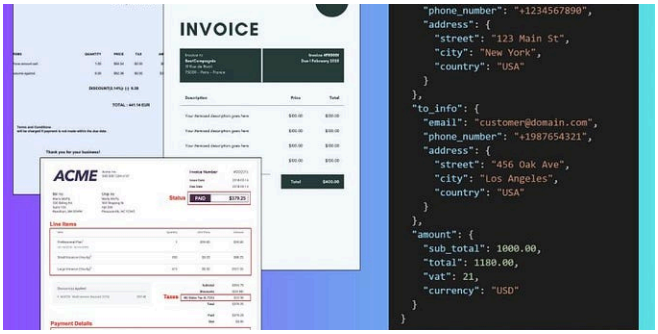


In Coding Nexus by Algo Insights

Google's 69-Page Prompt Engineering Masterclass: What's...

I've been writing about tech for three years, and let me tell you, nothing's grabbed me...

Apr 13 178 8

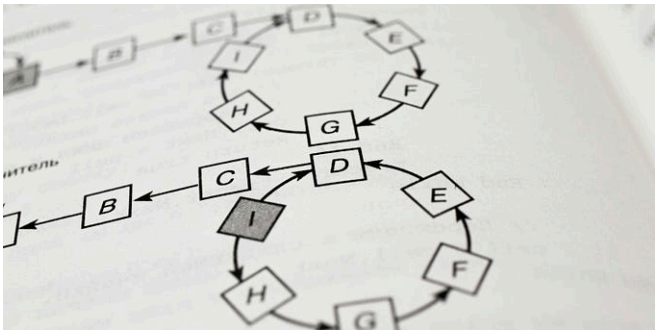


In Towards AI by Jeremy Arancio

Deploy an in-house Vision Language Model to parse millions...

How to build a Document Parsing Pipeline to process millions of documents using Qwen-...

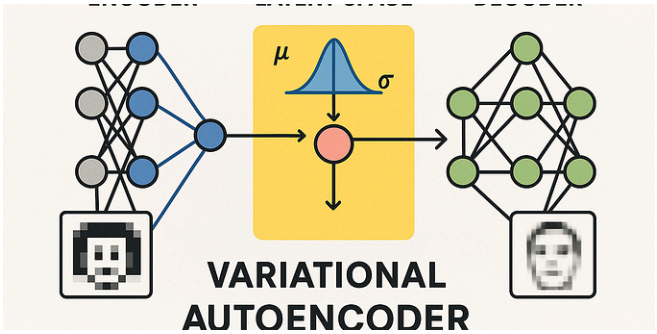
Apr 23 1K 16



In AI monks.io by JIN

Beyond Algorithms: Embracing Architecture + Data + Compute

Why Model Topologies and Massive Scale, Not Novel Math, Are Powering Modern AI



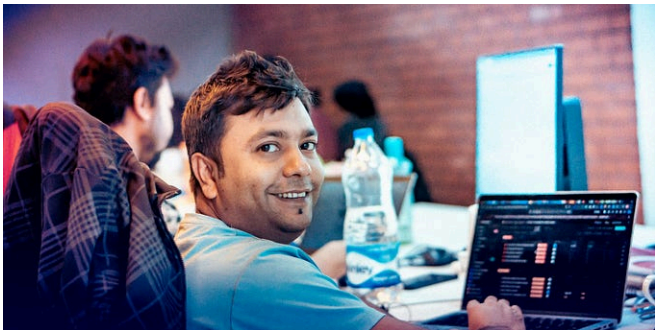
In T3CH by Francesco Franco

Implementing a Variational Autoencoder with Keras

In my previous blog post, we studied the concept of a Variational Autoencoder (or VA...

6d ago 120 2

6d ago 152 2



Analyst Uttam

In Generative AI by Mil Hoornaert

This SQL Trick Cut My Query Time by 80%

🚪 The Problem That Wasted My Hours (And Sanity)

Apr 21 589 32

Stop Using External AIs Like ChatGPT, Run A Local AI Server

Why I'm doing it and how you can do it too

Apr 10 901 20

See more recommendations