MET CS 767 Assignment 6: Bayesian Networks

Brendan Torok

1. Requirements for your application

My application will use a Regime-Switching Dynamic Bayesian Network (RS-DBN) to solve a problem of reconciling fast moving market data with slow moving snapshot company fundamental information. This application will determine whether a stock is entering a bull, bear, or sideways regime based on the data distribution with a fundamental twist. Strong or weak fundamental stock metrics increase the 'stickiness' of the bull or bear state respectively. When the data is run through the model, a strong buy signal will be indicated by a short-term drop in price with strong fundamentals and a prior bull regime. The probability of still being in a bull regime remains high despite the short-term drop.

Step 1, Parameterization:

At a high level, the regime $S_t \in$ {bull, bear, sideways} is hidden so only market features (x_t) and fundamentals (f) are observable. This application will learn $P(S_t \mid S_{t-1}, f)$. Given fundamentals, we will compute the transition probabilities $P(S_t = s' \mid S_{t-1} = s, f) = \text{softmax}_{s'}(ß_s{}^T{}_{,s'})$

Where:

- f = vector of fundamental metrics (valuation, profitability, etc..)
- $ß_s{}^T{}_{,s'}$ = parameters to learn
- softmax ensures probabilities across s´ sum to 1.

Step 2, Inference of Hidden Regimes:

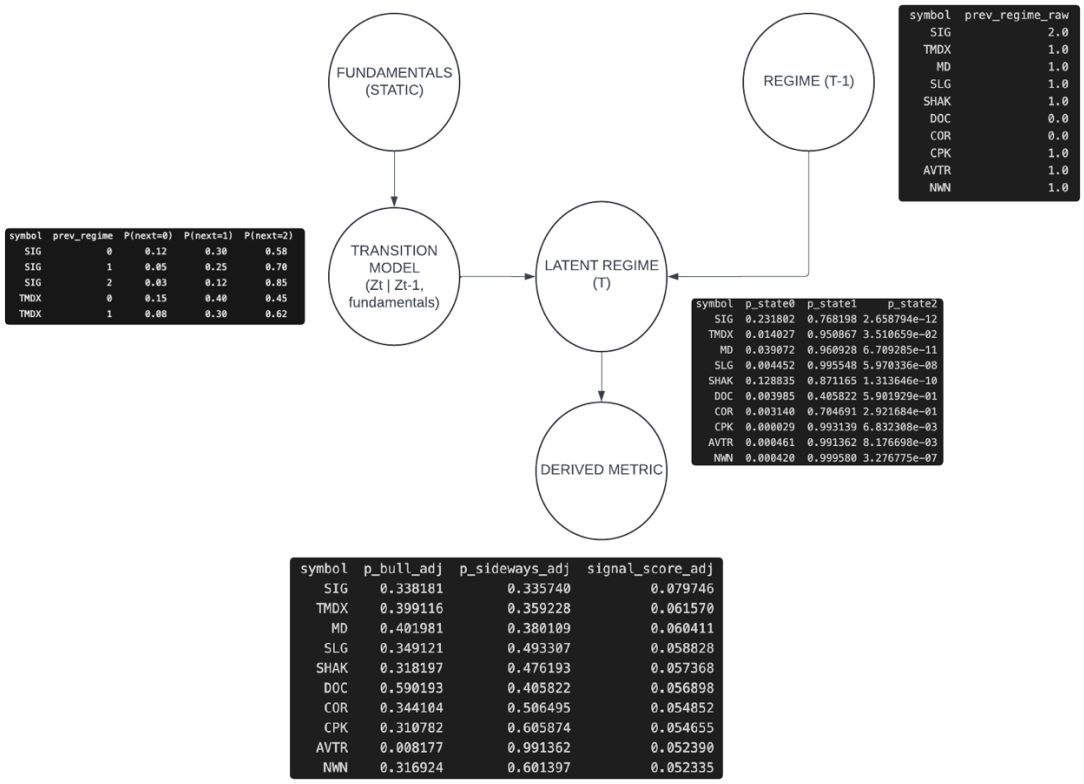
Regimes are not labeled so unsupervised learning is done with some initial guess for the emission parameters controlling how each regime generates observed returns/volatility and transition parameters controlling $ß_s{}^T{}_{,s'}$ in the softmax. For each stocks time series, a forward-backward algorithm is run to compute the posterior probability that the stock is in the regime at a given time and the posterior probability of regime transition between a given time.

Step 3, Updating Transition Parameters:

The transition parameters are updated to maximize the log-likelihood of transitions using weighted logistic regression. Gradient descent is used in this step, alternating between step 3 and step 2 until convergence. The end result is that the transition probabilities are tied to fundamentals so strong fundamentals gives high probability of staying in or moving into bull/sideways regime, and weak fundamentals high a higher probability of staying in or moving into the bear regime.

The two main inputs for each stock are the fast-moving market data and static fundamental data. The daily data consists of 1 year of daily market features such as relative volume changes, overnight gap price changes, log price ranges, and log returns. The fundamentals consist of data such as distance to moving averages, volatility, momentum, profitability, leverage, and growth metrics. The RS-DBN produces the outputs of the current regime posterior which is a probability distribution over regimes at the most recent time step indicating if the stock is currently in bull/bear/sideways regime. An output of the transition probabilities is taken for short-term forecasting how likely a move into each regime is in the next time period. A final output of a derived trading signal is given based on combining the most recent price action, the posterior probability, and transition probability. This final signal score indicates stocks with recent price drops that are less likely to enter a regime change, therefore they are more likely to reverse the recent price drop trend and return to their regular regime. In other words, this application will identify stocks with short-term negative trends that are most likely to reverse and gain value based on their historical price action and current fundamentals.

2. Diagram



symbol	prev_regime_raw
SIG	2.0
TMDX	1.0
MD	1.0
SLG	1.0
SHAK	1.0
DOC	0.0
COR	0.0
CPK	1.0
AVTR	1.0
NWN	1.0

symbol	prev_regime	P(next=0)	P(next=1)	P(next=2)
SIG	0	0.12	0.30	0.58
SIG	1	0.05	0.25	0.70
SIG	2	0.03	0.12	0.85
TMDX	0	0.15	0.40	0.45
TMDX	1	0.08	0.30	0.62

symbol	p_state0	p_state1	p_state2
SIG	0.231802	0.768198	2.658794e-12
TMDX	0.014027	0.950867	3.510659e-02
MD	0.039072	0.960928	6.709285e-11
SLG	0.004452	0.995548	5.970336e-08
SHAK	0.128835	0.871165	1.313646e-10
DOC	0.003985	0.405822	5.901929e-01
COR	0.003140	0.704691	2.921684e-01
CPK	0.000029	0.993139	6.832308e-03
AVTR	0.000461	0.991362	8.176698e-03
NWN	0.000420	0.999580	3.276775e-07

symbol	p_bull_adj	p_sideways_adj	signal_score_adj
SIG	0.338181	0.335740	0.079746
TMDX	0.399116	0.359228	0.061570
MD	0.401981	0.380109	0.060411
SLG	0.349121	0.493307	0.058828
SHAK	0.318197	0.476193	0.057368
DOC	0.590193	0.405822	0.056898
COR	0.344104	0.506495	0.054852
CPK	0.310782	0.605874	0.054655
AVTR	0.008177	0.991362	0.052390
NWN	0.316924	0.601397	0.052335

3. Example 1

Input:

The model uses core daily market microstructure features of log_ret, overnight_gap, log_range, and rel_vol for each stock in the S&P1500. These features are treated as emission variables in a

Gaussian Hidden Markov Model (HMM) where the latent regimes (bull, sideways, bear) are unobserved.

symbol	date	log_ret	overnight_gap	log_range	rel_vol
ZWS	2025-11-28	-0.008974	0.006833	0.016220	0.648874
ZWS	2025-12-01	0.000838	-0.011173	0.021506	0.833960
ZWS	2025-12-02	0.001674	0.005015	0.012602	1.170634
ZWS	2025-12-03	-0.007767	0.001254	0.016730	0.960977
ZWS	2025-12-04	-0.011233	-0.000632	0.018532	1.102343

Process:
The emission features are standardized to ensure numerical stability and balanced influence across variables. The HMM is initialized using empirical slices of data corresponding to low, medium, and high returns to improve convergence. The model estimates posterior regime probabilities for each day and stock.

Output:
A short-term mean-reversion signal is constructed by combining the posterior probabilities of the bull and sideways regimes and weighting them by the magnitude of a negative daily return. This signal identifies stocks where a recent price drop is unlikely to represent a regime shift and is therefore more likely to be temporary. The resulting signal score is used for relative ranking rather than probabilistic interpretation and is further refined downstream in the RS-DBN framework where the posterior probabilities are adjusted with transition probabilities.

	symbol	log_ret	p_bull	p_sideways	signal_score
248261	OLN	-0.048231	4.641672e-12	0.969129	0.046742
114003	ENOV	-0.046130	7.461285e-12	0.966444	0.044582
341728	URBN	-0.044676	3.146695e-07	0.982800	0.043907
212836	MD	-0.043886	6.709285e-11	0.960928	0.042171
367375	WYNN	-0.042636	2.499703e-07	0.968084	0.041275
68338	CHH	-0.042341	1.612213e-11	0.965542	0.040882
302414	SLG	-0.040593	5.970336e-08	0.995548	0.040412
23203	ANF	-0.040462	6.507342e-09	0.991723	0.040127
192433	KOP	-0.040180	1.697122e-08	0.993350	0.039913
42193	BDN	-0.039944	4.405570e-06	0.996635	0.039810

4. Example 2

Input:
The input here is the posterior regime probabilities produced by the Gaussian HMM and a short-term price movement feature (drop_3d) which captures recent price declines.

symbol	date	log_ret	drop_3d	p_state0	p_state1	p_state2
SIG	2025-12-04	-0.045975	-0.118332	0.231802	0.768198	2.658794e-12
TMDX	2025-12-04	-0.010115	-0.081190	0.014027	0.950867	3.510659e-02
MD	2025-12-04	-0.043886	-0.077243	0.039072	0.960928	6.709285e-11
SLG	2025-12-04	-0.040593	-0.069832	0.004452	0.995548	5.970336e-08
SHAK	2025-12-04	-0.037842	-0.072216	0.128835	0.871165	1.313646e-10
DOC	2025-12-04	-0.006382	-0.057126	0.003985	0.405822	5.901929e-01
COR	2025-12-04	0.006895	-0.064487	0.003140	0.704691	2.921684e-01
CPK	2025-12-04	-0.016922	-0.059625	0.000029	0.993139	6.832308e-03
AVTR	2025-12-04	-0.012478	-0.052414	0.000461	0.991362	8.176698e-03
NWN	2025-12-04	-0.035084	-0.056990	0.000420	0.999580	3.276775e-07

Process:
A multinominal logistic regression model is trained to estimate regime transition probabilities using slow-moving fundamental characteristics of each stock. The model conditions transition on both previous regime and firm fundamentals, which allows for stronger fundamentals to increase regime persistence and reduce the likelihood of adverse transitions. For each stock a fundamental-conditioned transition matrix is constructed. The current HMM posterior vector is propagated forward through the matrix to obtain adjusted regime probabilities which produces a regime probability that integrates with recent market behavior and longer-term fundamental context.

Output:
The adjusted bull and sideways probabilities are combined with the recent price decline to compute a final RS-DBN signal score. Stocks with high scores represent cases where recent price weakness is unlikely to trigger a regime transition, and is therefore more likely to reverse.

symbol	drop_3d	p_bull_adj	p_sideways_adj	signal_score_adj
SIG	-0.118332	0.338181	0.335740	0.079746
TMDX	-0.081190	0.399116	0.359228	0.061570
MD	-0.077243	0.401981	0.380109	0.060411
SLG	-0.069832	0.349121	0.493307	0.058828
SHAK	-0.072216	0.318197	0.476193	0.057368
DOC	-0.057126	0.590193	0.405822	0.056898
COR	-0.064487	0.344104	0.506495	0.054852
CPK	-0.059625	0.310782	0.605874	0.054655
AVTR	-0.052414	0.008177	0.991362	0.052390
NWN	-0.056990	0.316924	0.601397	0.052335

5. Scaling

One initial obstacle of this code is the reliance on static fundamental stock information. In a real world application fundamental stock metrics should be available for every trading day. Even for metrics that are only updated quarterly, the dataset should contain up to date fundamentals for every quarter. Additionally, the way this program is written makes it very difficult to maintain in a large-scale business where scripts must be quickly executable. In order to make this more maintainable the structure should change to a boiler-plate OOP implementation and wrapper methods to allow for quick execution of the script through APIs if the data is requested. Further improvements would be to present the data in a way that is more easily interpretable, and instead of a static dataset, the data should be updated daily by an additional helper script.

Other obstacles include a heavy memory requirement of this code by reading in a wide CSV and converting it to a long panel. If this CSV format was used for a larger dataset containing years of data along with yearly quarterly fundamental statistics, it would quickly lead to an OOM. This implementation would have to be changed to store real-time and statistical data in a cloud database in the ‘long’ format which could be retrieved by the algorithm as needed.

An additional constraint of this implementation is the requirement for bottlenecking memory heavy loops to be executed in sequence. This would result in very long running times for larger datasets and eventually become unusable. The implementation should instead be changed to use distributed computing where the large tasks can be divided into smaller subtasks ad processed

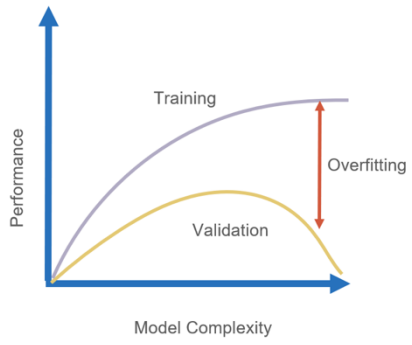
simultaneously over a network of multiple computers all acting on the same database. This would require changing a large amount of the database functions to use big data concepts map reduce, data replication, partitioning, and load balancing.

The biggest issue with the current implementation however is the look-ahead bias present due to using fit_transform and scaling on the entire daily dataset instead of using a rolling window, or a recursive scaler. When the scaler is fit only on a rolling window or on past data, then is used to transform the current days data it avoids data leakage. Similarly, the HMM model fits the model on the entire model which results in the hidden states from 5 years ago being influenced by the current market structure. For a real-world application this implementation should be changed to use a rolling training window where the dataset is progressively rebuilt and the model is retrained after every new window is trained on.

6. Alternatives

This current application would be more difficult to implement with both a neural network or genetic algorithm. The regime-switching prediction task done here with an unlabeled dataset lends itself very well to a RS-DBN implementation, whereas successfully identifying 'hidden' market regimes without labeled data is complex. To apply a similar algorithm with a neural network, first some work would need to be done into determining how to label the data for each day which is a difficult task in and of itself. Additionally, given the current dataset size is only a few thousand datapoints, it is likely not enough data to train an effective neural network dataset as neural networks typically require significantly more data to learn temporal dependencies without memorizing market noise. Another complication of neural networks is that a neural network that is trained on a period of the market that has low volatility might over-optimize and completely fail in alternative market conditions whereas the simpler implementation of the RS-DBN may generalize better, a good reminder that increased model complexity does not mean increased model performance.

For genetic algorithms (GA), fitness must be defined as profitability of a trading strategy based on the regimes, but the GA would likely begin to overfit specific rules that made money in the past but capture noise rather than true economic regimes. Additionally, the required simulation of thousands of generations in the algorithm would require a full backtest of the entire history for every candidate model—resulting in an even larger computational task compared to the fast matrix algorithm of the current implementation. Perhaps the best implementation would be to use a GA to tune the HMM in the model to find the optimal number of days for some rollup data features like momentum and volatility.



7. References

Appendix 1:
Link to chat: https://chatgpt.com/share/6937bc04-3ee0-8006-aa92-7393c0eeae43
Link to github repo: https://github.com/btorok-bu/METCS767_hw6

Appendix 2:

Code to train Gaussian HMM:

```
emission_cols = ["log_ret", "log_range", "rel_vol", "overnight_gap"]

# remove all rows with any NaN values in the emission features
mask = panel_df[emission_cols].notna().all(axis=1)
panel_hmm = panel_df[mask].copy()

# build X and lengths from panel_hmm
X_raw = panel_hmm[emission_cols].values

# standardize the values
scaler_hmm = StandardScaler()
X = scaler_hmm.fit_transform(X_raw)

lengths = (
    panel_hmm.groupby(ID_COL)["date"]
    .size()
    .sort_index()
    .to_list()
)

# to speed up convergence we can initialize means of log_ret instead of giving HMM
random init

panel_hmm['log_ret_std'] = X[:, 0] # first column is scaled log_ret

panel_hmm_sorted = panel_hmm.sort_values('log_ret_std')
n = len(panel_hmm_sorted)
third = n //3

# get the three slices
bear_slice = panel_hmm_sorted.iloc[:third]
sideways_slice = panel_hmm_sorted.iloc[third:2*third]
bull_slice = panel_hmm_sorted.iloc[2*third:]

# build initial means in standardized feature space
X_df = pd.DataFrame(X, index=panel_hmm.index, columns=emission_cols)
mu_bear = X_df.loc[bear_slice.index].mean().values
mu_side = X_df.loc[sideways_slice.index].mean().values
```

```
mu_bull = X_df.loc[bull_slice.index].mean().values

init_means = np.vstack([mu_bear, mu_side, mu_bull])

# get covariances estimated from slice
def cov_from_slice(idx):
    X_s = X_df.loc[idx].values
    return np.cov(X_s.T) + 1e-6 * np.eye(X_s.shape[1])

cov_bear = cov_from_slice(bear_slice.index)
cov_side = cov_from_slice(sideways_slice.index)
cov_bull = cov_from_slice(bull_slice.index)

# initialize covariates
init_covars = np.stack([cov_bear, cov_side, cov_bull], axis=0)

# fit 3-state Gaussian HMM
hmm_model = GaussianHMM(
    n_components=3,
    covariance_type="full",
    n_iter=200,
    min_covar=1e-5,
    init_params='st', # don't re-init means or covars since we set them
    params='stmc',
    random_state=123
)

hmm_model.means_ = init_means
hmm_model.covars_ = init_covars
hmm_model.fit(X, lengths)

# posterior probabilities P(state k | observations)
posteriors = hmm_model.predict_proba(X)
states = hmm_model.predict(X)  # Viterbi state sequence

panel_hmm["regime_raw"] = states
panel_hmm[["p_state0", "p_state1", "p_state2"]] = posteriors

hmm_cols = ['regime_raw', "p_state0", "p_state1", "p_state2"]
# drop duplicate cols from panel df if they exist from previous run
panel_df = panel_df.drop(columns=hmm_cols, errors='ignore')

# merge HMM outputs back onto full panel_df by (symbol, date)
panel_df = panel_df.merge(
    panel_hmm[[ID_COL, "date", "regime_raw", "p_state0", "p_state1", "p_state2"]],
    on=[ID_COL, "date"],
    how="left",
)
```

Code to run multinominal logistic regression:

```
# now run multinominal logistic regression on new dataframe to predict regime_raw from
prev_regime_raw

X_fund = transitions_df_clean[fundamental_cols]
prev_regime = transitions_df_clean['prev_regime_raw'].astype(int)
next_regime = transitions_df_clean['regime_raw'].astype(int)

# scale the fundamental data
scaler = StandardScaler()
X_fund_scaled = scaler.fit_transform(X_fund)

# get the onehot values of prev regime
prev_regime_onehot = pd.get_dummies(prev_regime, prefix='prev_regime', dtype=int)
X_all = np.hstack([X_fund_scaled, prev_regime_onehot.values])

# add class weights as bull case is rare
classes = np.unique(next_regime)
class_weights = dict(
    zip(classes, compute_class_weight('balanced', classes=classes, y=next_regime))
)

# now run multinominal logistic regression
logit_trans = LogisticRegression(
    multi_class='multinomial',
    penalty='elasticnet', # encourages group sparsity, stabilizing transitions
    solver='saga', # required for elastic net
    l1_ratio=0.2,
    C=0.3, # stronger regularization than usual to reduce regime-flip noise
    class_weight=class_weights,
    max_iter=2000,
    random_state=123,
)

logit_trans.fit(X_all, next_regime)
# get the states/class order used by logistic regression
logit_classes = logit_trans.classes_
n_states = len(logit_classes)
```

Code to use transition matrix and HMM posterior probabilities to get adjusted probabilities

```
# map state index -> position in class_ array
```

```
state_to_classpos = {int(s): int(i) for i, s in enumerate(logit_classes)}

def compute_adjusted_probs_row(row):
    # if fundamentals missing, fall back to HMM probabilities
    if row[fundamental_cols].isna().any():
        out = {}
        for label, state in label_to_state.items():
            out[f"p_{label}_adj"] = row.get(f"p_{label}", np.nan)
        return pd.Series(out)

    # scale the fundamental columns
    f_vec = row[fundamental_cols].values.reshape(1, -1)
    f_scaled = scaler.transform(f_vec)

    # build a fundamental-conditioned transition matrix
    A = np.zeros((n_states, n_states))

    # one-hot basis for prev_regime in classes_ order
    eye = np.eye(n_states, dtype=float)

    for i, prev_state in enumerate(logit_classes):
        one_hot_prev = eye[i].reshape(1, -1) # length of n_states
        X_row = np.hstack([f_scaled, one_hot_prev])
        probs_next = logit_trans.predict_proba(X_row)[0]

        A[int(prev_state), :] = probs_next

    # get the current HMM posterior pi_t in state-index order
    pi_t = np.zeros(n_states)
    for s in logit_classes:
        pi_t[state_to_classpos[int(s)]] = row.get(f'p_state{int(s)}', 0.0)


    pi_next = pi_t @ A

    out = {}
    for label, state in label_to_state.items():
        idx = state_to_classpos[int(state)]
        out[f"p_{label}_adj"] = pi_next[idx]

    return pd.Series(out)
```

Code to return top candidate stocks:

```
# compute final score and print top candidates
```

```
latest_date = panel_with_fund['date'].max()

# get latest slice to be stocks with drop in past 3 days
latest_slice = panel_with_fund.loc[
    (panel_with_fund['date'] == latest_date)
    & panel_with_fund['drop_3d'].notna()
    & panel_with_fund['p_bull'].notna()
].copy()

# only consider names where price fell in the last 3 days
latest_slice = latest_slice[latest_slice['drop_3d'] < 0]

adj_probs = latest_slice.apply(compute_adjusted_probs_row, axis=1)
latest_slice = pd.concat([latest_slice, adj_probs], axis=1)

# compute final RS-DBD score:
latest_slice['signal_score_adj'] = (
    (latest_slice['p_bull_adj'] + latest_slice['p_sideways_adj'])
    * (-latest_slice['drop_3d'])
)

# rank by adjusted score
latest_slice = latest_slice.sort_values('signal_score_adj', ascending=False)

top_candidates = latest_slice[
    [ID_COL,
    'drop_3d',
    'p_bull_adj', 'p_sideways_adj',
    'signal_score_adj']
]

# print the top 10 companies
print(top_candidates.head(10).to_string(index=False))
```