GROUP WORK PROJECT # _2__ MSCFE 660: RISK MANAGEMENT
GROUP NUMBER: ________9039____

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Christopher Enny Ofikwu	Nigeria	c.ofikwu@outlook.com	
Bharat Swami	India	bharatswami1299@gmail.com	
Chengjie Wang	China	cjay-wang@hotmail.com	

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).	
Team member 1	**Christopher Enny Ofikwu**
Team member 2	**Bharat Swami**
Team member 3	**Chengjie Wang**

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed.
Note: You may be required to provide proof of your outreach to non-contributing members upon request.

N/A

Step 1: Implementing Hidden Markov Model (HMM) Algorithms

(a). Student A: Forward and Backward Algorithms

Forward Algorithm Pseudocode (computes the probability of observing a sequence given the model):

```
FUNCTION Forward_Algorithm(observations, states, start_prob, trans_prob, emit_prob):
    T = length(observations)
    N = number of states
    alpha = matrix[T, N]  // Forward probabilities

    // Initialization
    FOR each state i:
        alpha[0, i] = start_prob[i] * emit_prob[i, observations[0]]

    // Recursion
    FOR t = 1 to T-1:
        FOR each state j:
            alpha[t, j] = 0
            FOR each state i:
                alpha[t, j] += alpha[t-1, i] * trans_prob[i, j]
            alpha[t, j] *= emit_prob[j, observations[t]]

    // Termination
    prob = sum(alpha[T-1, :])
    RETURN alpha, prob
```

BACKWARD ALGORITHM PSEUDOCODES

```
FUNCTION Backward_Algorithm(observations, states, trans_prob, emit_prob):
    T = length(observations)
    N = number of states
    beta = matrix[T, N]  // Backward probabilities

    // Initialization
    FOR each state i:
        beta[T-1, i] = 1

    // Recursion
    FOR t = T-2 down to 0:
        FOR each state i:
            beta[t, i] = 0
            FOR each state j:
                beta[t, i] += beta[t+1, j] * trans_prob[i, j] * emit_prob[j, observations[t+1]]

    // Termination
    prob = 0
    FOR each state i:
        prob += beta[0, i] * start_prob[i] * emit_prob[i, observations[0]]
    RETURN beta, prob
```

Toy Example:

- States: {Bull, Bear}
- Observations: {Up, Down}
- Start Probabilities: P(Bull) = 0.6, P(Bear) = 0.4
- Transition Probabilities: P(Bull→Bull) = 0.7, P(Bull→Bear) = 0.3, P(Bear→Bull) = 0.4, P(Bear→Bear) = 0.6
- Emission Probabilities: P(Up|Bull) = 0.8, P(Down|Bull) = 0.2, P(Up|Bear) = 0.3, P(Down|Bear) = 0.7
- Observation Sequence: [Up, Down]

Run the forward algorithm:

1. Initialize: $\alpha(0, \text{Bull}) = 0.6 * 0.8 = 0.48$, $\alpha(0, \text{Bear}) = 0.4 * 0.3 = 0.12$
2. Recursion for t=1 (Down): Compute $\alpha(1, \text{Bull})$ and $\alpha(1, \text{Bear})$ using the formula.
3. Sum $\alpha(1, :)$ for the total probability.

The backward algorithm follows similarly, starting from the end.

(b). Student B: Backward Viterbi Algorithm

Viterbi Algorithm Pseudocode (finds the most likely sequence of hidden states):

```
FUNCTION Viterbi_Algorithm(observations, states, start_prob, trans_prob, emit_prob):
    T = length(observations)
    N = number of states
    V = matrix[T, N]  // Probability of most likely path
    path = matrix[T, N]  // Backpointers

    // Initialization
    FOR each state i:
        V[0, i] = start_prob[i] * emit_prob[i, observations[0]]
        path[0, i] = 0

    // Recursion
    FOR t = 1 to T-1:
        FOR each state j:
            V[t, j] = 0
            FOR each state i:
                prob = V[t-1, i] * trans_prob[i, j] * emit_prob[j, observations[t]]
                IF prob > V[t, j]:
```

```
                V[t, j] = prob
                path[t, j] = i

    // Termination
    best_path_prob = max(V[T-1, :])
    best_last_state = argmax(V[T-1, :])

    // Backtracking
    best_path = [best_last_state]
    FOR t = T-1 down to 1:
        best_last_state = path[t, best_last_state]
        best_path.append(best_last_state)
    best_path.reverse()

    RETURN best_path, best_path_prob
```

Toy Example:

Using the same setup as above, apply the Viterbi algorithm to find the most likely state sequence (e.g., [Bull, Bear]) for the observation sequence [Up, Down].

(c). Student C: Baum-Welch Algorithm

Baum-Welch Algorithm Pseudocode (parameter estimation for HMM):

```
FUNCTION Baum_Welch(observations, states, init_start_prob, init_trans_prob, init_emit_prob,
max_iters):
    T = length(observations)
    N = number of states

    FOR iter = 1 to max_iters:
        // E-step: Compute forward and backward probabilities
        alpha, _ = Forward_Algorithm(observations, states, start_prob, trans_prob, emit_prob)
        beta, _ = Backward_Algorithm(observations, states, trans_prob, emit_prob)

        // Compute xi and gamma
        xi = 3D_matrix[T-1, N, N]  // Probability of transitioning from i to j at t
        gamma = matrix[T, N]  // Probability of being in state i at t

        FOR t = 0 to T-2:
            denom = 0
            FOR i, j in states:
                denom += alpha[t, i] * trans_prob[i, j] * emit_prob[j, observations[t+1]] * beta[t+1, j]
            FOR i, j in states:
                xi[t, i, j] = (alpha[t, i] * trans_prob[i, j] * emit_prob[j, observations[t+1]] * beta[t+1, j])
```

```
/ denom
                gamma[t, i] += xi[t, i, j]

        // M-step: Update parameters
        FOR i in states:
            start_prob[i] = gamma[0, i]
            FOR j in states:
                trans_prob[i, j] = sum(xi[:, i, j]) / sum(gamma[:, i])
            FOR obs in observations:
                emit_prob[i, obs] = sum(gamma[t, i] where observations[t] == obs) / sum(gamma[:, i])

    RETURN start_prob, trans_prob, emit_prob
```

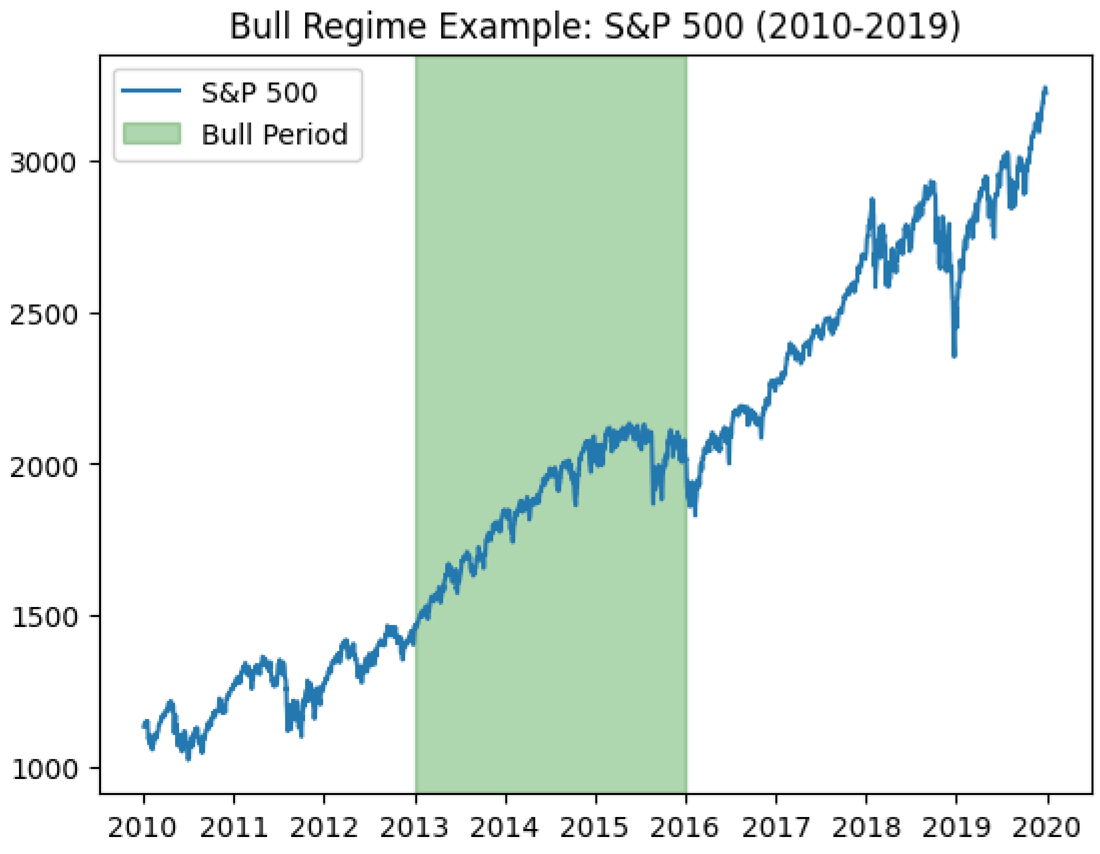
Step 2: Identifying Market Regimes

Student A: Identify and illustrate examples of bull regimes.

Financial market assets experience lasting upward movement when investors hold strong confidence while economic signals remain favorable. This bull market condition becomes known as a bull regime. The market consistently experiences price increase for stocks and estates while other investments rise from multiple economic factors including company profitability combined with dynamic economic policies alongside low interest rates. The atmosphere of positive investor sentiment grows across markets which results in rising purchasing behavior together with increased deal frequency and investors entering riskier investment sectors. Confidence in the economy mainly resides on fundamental macroeconomic conditions such as GDP expansion combined with fewer unemployed people and steady consumer market spending and modest inflation levels. Different parts of the economy expand through increased company IPOs and intensified merger and acquisition activity. The fundamental upward trend of the market resists immediate temporary declines and remains positive in the long term. When bull markets extend in duration speculative events may occur that confuse genuine fundamental-driven growth from overheated artificial asset bubbles. Central banks and policymakers support economic expansion by implementing policies but must maintain continuous monitoring to prevent overinflated risks.

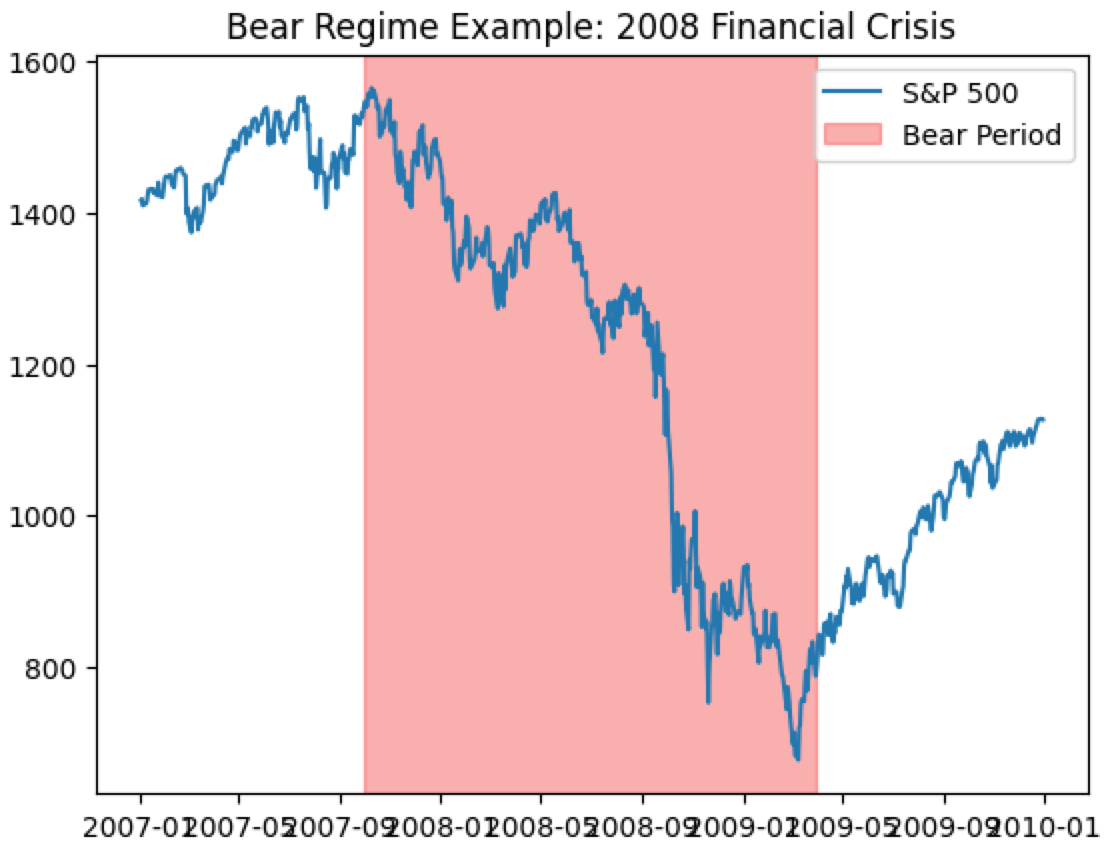
For example, The U.S. stock market from 2010–2019 (post-financial crisis recovery). S&P 500 grew ~250% with low volatility

The graph below illustrates and highlight periods of sustained upward trends for S&P500 (e.g., 2013–2015).



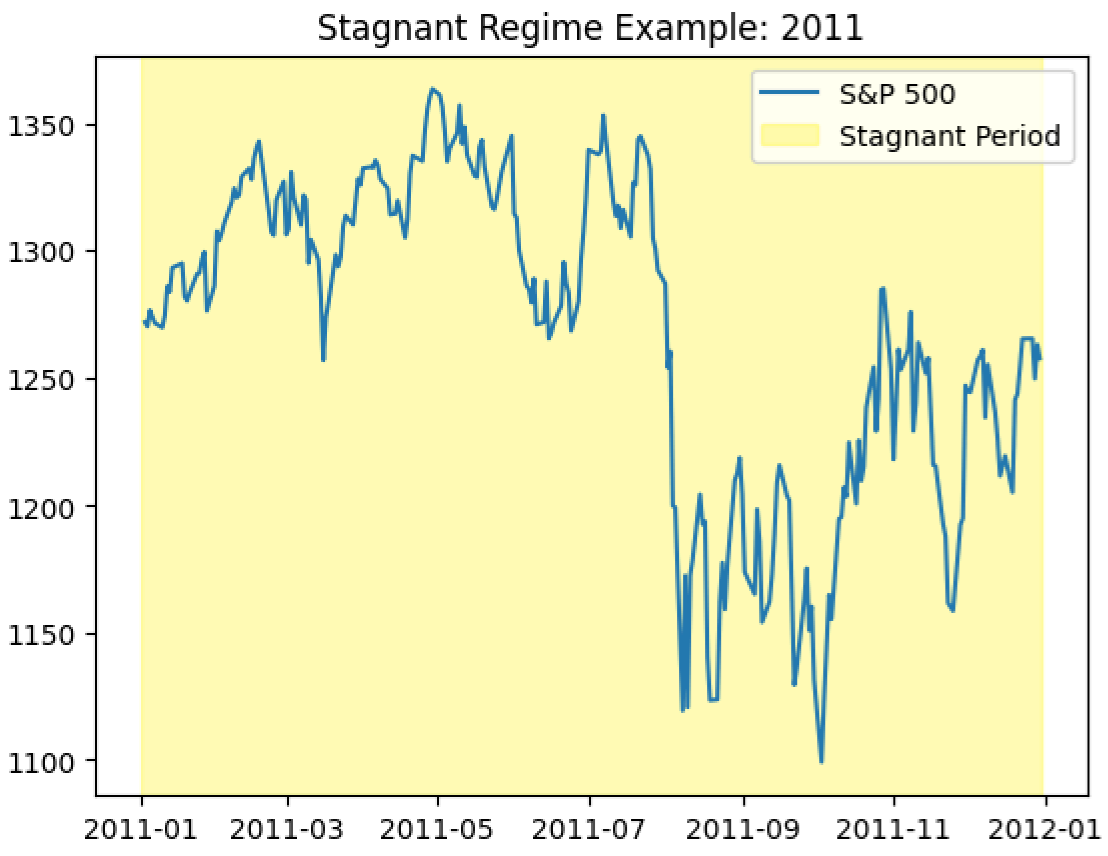
(b). Student B: Bear Regimes

The bear market shows itself through persistent price reductions of assets with increased market turmoil along with broad investor negativity because of worsening economic situations. These market conditions show the effects of systemic issues including unemployment rise and economic contraction and financial instability. Weak corporate performance combines with declining real estate values and falling stock prices in such situations. The 2008 global financial crisis served as a perfect example of this regime where the U.S. housing market collapsed alongside major institutions like Lehman Brothers to cause the S&P 500 index to drop 50% between October 2007 and March 2009. Panicked investors responded with urgent asset sales which coupled with liquidity problems and forced them into buying Treasury bonds and gold while selling both stocks and other financial products. The panicked stock market triggered an extreme volatility surge that sent the VIX "fear index" to record-breaking levels at the same time credit markets shut down to worsen economic downturn. The spread of negative feelings throughout households and enterprises generated decreased consumer spending and terminated investments which ultimately caused an international economic downturn. During bear markets governments combined with central banks implemented extraordinary intervention methods and quantitative easing programs to support market stability which confirmed the basic and systemwide dangers in bear market lengths.



(c). Student C: Stagnant Regimes

Stagnant financial market regimes occur during periods with small price fluctuations and limited market uncertainty which indicates investor indecision about future trends. A stagnant market regime develops we find in uncertain periods and after substantial economic or political changes when investors exercise caution due to confusing market conditions or lack of fresh developments. These market conditions present stock prices that avoid major upward or downward movement while staying confined to a limited trading zone always between two boundaries. Although challenging for traders who wish to spot investment possibilities, it provides stability for investors looking at extended time horizons. The low variation of prices indicates minor market swings while little movement demonstrates investors lack direction regarding market performance thus stabilizing the status quo.



STEP 3

Hidden Markov Models (HMMs)

The Hidden Markov Model represents a statistical tool that describes processes having hidden states which exhibit Markov processes. Although hidden state sequences remain invisible the observed emissions or output exists and depends on the current hidden state.

Formally, an HMM is defined by the following components:

(a) States $Q = \{q_1, q_2, \ldots\ldots, q_N\}$: A finite set of hidden states through which the system transitions over time.

(b) Observation symbols å = $\{s_1, s_2, \ldots.., s_M\}$: A finite set of observable symbols emitted from the states.

(c) Initial state distribution p = {p}, where p = P(q_1 = qi): The probability of starting in state q_i.

(d) State transition probabilities $A = \{a_{ij}\}$, where $a_{ij} = P(q_{t+1} = q \mid q_t = q_i)$: The probability of transitioning from state qi to state q_j.

(e) Emission probabilities $B = \{b_i(o_k)\}$, where $b_i(o_k) = P(o_k \mid q_i)$: The probability of observing symbol o_k given that the system is in state q_i.

These parameters can be compactly denoted as l = (p, A, B).

The model assumes:

1. The process demonstrates the Markov property because the probabilities for transitioning to following states depend only on present conditions and ignore previous states.

2. The hidden states Q make observations at time t independent from previous states; thus each observation probability depends solely on the current hidden state.

HMMs serve as a widespread tool for time-series modeling and sequential data analysis which finds practical applications in finance through regime switching models as well as speech recognition together with bioinformatics and natural language processing.

There are three canonical problems associated with HMMs:

1. Evaluation: Given a model (*lambda)* and a sequence of observations *O*, compute the probability *P(O* | *lambda)*. This is efficiently solved using the Forward Algorithm.

2. Decoding: Given *lambda* and *O*, determine the most likely sequence of hidden states that led to *O*. The Viterbi Algorithm is typically used for this purpose.

3. Learning: Given *O* and a general model structure, adjust *lambda* to maximize *P(O | lambda)*. This is addressed using the Baum-Welch Algorithm, an instance of the Expectation-Maximization (EM) algorithm.

By decoupling the observed data from the underlying generative state process, HMMs allow for modeling systems with rich temporal dynamics while managing uncertainty about the underlying structure.

STEP 4

Student A Write-up: Macro Research, Indicator Identification, Dataset Retrieval, and Data Cleaning

The design approach for a probabilistic graphical model to forecast oil prices receives its foundation through research and data engineering activities during the initial phases. The methodology includes studying macroeconomic data first and determining necessary indicators before completing a data retrieval and cleaning process.

Macro Research

Strategies for global macroeconomics heavily rely on crude oil as a fundamental commodity. Multiple elements affect crude oil prices including market tension dynamics and energy policy decisions and supply and demand patterns together with economic output measures and industrial manufacturing data. The development of an accurate market model requires an understanding of all these market influences.

Oil trading activities on worldwide markets use U.S. dollar currency pricing which requires knowledge of economic data mainly from the United States and China and OECD member states. The U.S. Energy Information Administration (EIA) together with Federal Reserve Economic Data (FRED) deliver trustworthy information at the macroeconomic scale.

Indicator Identification From the EIA, physical market indicators include:

1. OPEC and Non-OPEC Supply
2. OECD and Non-OECD Demand
3. Inventories and Spare Capacity

From FRED, macroeconomic indicators include:

1. Consumer Price Index for Energy (CPIENGSL)
2. Industrial Production Index (INDPRO)
3. U.S. Treasury Yield Curve
4. Global inflation and interest rates

The selected indicators serve as markers because they demonstrate advanced influence on oil values while providing relevant information frequently.

4. Dataset Retrieval Retrieval of relevant time-series data is automated using Python libraries:

1. eia-python for EIA datasets
2. fredapi for FRED datasets

Each API call involves:

1. Supplying a valid API key
2. Calling the appropriate series ID
3. Receiving JSON or DataFrame objects with time-indexed data

For example:

```
from fredapi import Fred
fred = Fred(‘api_key’)
cpi_data = fred.get_series('CPIENGSL')
```

5. Data Cleaning The raw datasets often contain issues such as:

1. Missing values
2. Inconsistent frequency or index formats
3. Redundant or irrelevant metadata

Using pandas, the process includes:

1. Forward or backward filling missing data
2. Aligning time indices across multiple datasets
3. Converting date formats
4. Resampling to ensure monthly frequency

Reusable functions (e.g., clean_EIA()) encapsulate these steps, promoting consistency.

```
def clean_EIA(df):
    df.index = pd.to_datetime(df.index)
    df = df.asfreq('M')
    df = df.fillna(method='ffill')
    return df.
```

Conclusion

Modeling needs effective macroeconomic research and data preparation as fundamental precursors. A clean and consistent dataset built from relevant indicators furnishes the probabilistic model with efficient groundwork.

Student B writes up to 5 pages that discuss the regime process and explain how the relevant code from hmms works.

In "Application of Probabilistic Graphical Models in Forecasting Crude Oil Price" Danish A.

Alvi uses Probabilistic Graphical Models (PGMs) to predict crude oil prices while focusing primarily on time-series discretization through regime detection. The transformation of continuous time-series data into discrete states becomes essential before using them as inputs in Bayesian Networks. The hidden Markov model identifies separate market regimes through its process known as hidden Markov model. These distinct market regimes become the basis for data discretization. In this section we provide detailed information about regime detection together with an explanation of code execution through the hmms Python library as demonstrated in the research thesis.

Regime Process Detection

Time-series data contains different "regimes" which can be detected using this analysis method for financial market information and other measurement sequences. The application of regime detection transforms continuous and complex oil price movements into three distinct states such as rising or falling or stable. The requirement for discrete variables exists because Bayesian Networks (as used in this thesis research) operate best with artifacts that have discrete variables for modeling probabilistic relationships.

The dissertation makes use of HMMs to conduct regime detection analysis. Probabilistic HMM models operate under the assumption that the targeted system functions through a Markov process having concealed states. A Markov property controls the state transitions while data emission occurs based on probability distributions from each hidden state. This application models hidden market states as specifically bullish or bearish or neutral regimes while using price variations or movement data as observable input.

STEPS IN REGIME DETECTION (Section 3.3.1, Page 42)

Data Preparation: The original time-series datasets sourced from the Energy Information Administration (EIA) and Federal Reserve Economic Data (FRED) consist of WTI Spot Price crude oil data and macroeconomic index data. Pandas library enables data cleaning and preprocessing of these datasets to maintain uniformity and manage absent values.

Differencing: Dynamic stationary can be achieved by applying differencing procedures on the time-series data. The calculation works by finding the variation between adjacent points in the test_data series (data_diff = test_data[series_id].diff()[1:]). Differencing allows researchers to monitor price alterations instead of studying fixed price measurements because it delivers stronger insights for regime detection.

Binary Encoding: The differentiating values from the original data follow a binary emissive sequence through which positive differences indicate 1 while non-positive decreases appear as 0. The observed data for this HMM takes the form of a binary sequence.

```
emit_seq = np.array(data_diff.apply(lambda x: 1 if x > 0 else 0).values)
```

HMM Application: An HMM processes the binary emission sequence by identifying the sequence of hidden states (regimes). A sequence of hidden states generates the observed binary sequence according to the HMM through probability emission rules. The Viterbi algorithm applies to locate hidden state sequences on the basis of observed data while determining the most probable sequences.

```
(log_prob, s_seq) = dhmm.viterbi(emit_seq)
```

Descritization output: The inferred discrete states exist in a new DataFrame named discrete_test and each column represents a time-series dataset as the values appear in three possible states (0, 1 or 2 which stand for distinct regimes). The Bayesian Network project utilizes the saved discrete data as an essential step for further modeling.

```
discrete_test[series_id] = s_seq

discrete_test.to_csv("./data/test_data.csv")
```

Purpose and Significance

Simplification: The conversion of continuous time-series data into discrete states makes Bayesian Networks support probability calculations through their discrete probabilistic framework.

Capturing Market Dynamics: The model reveals different market conditions (bullish or bearish trends) through regime identification because these conditions emerge based on macroeconomic and geopolitical influences.

Input Bayesian Network: The Bayesian Network uses discretized states to receive inputs and enables the model to determine relationships between macroeconomic factors and crude oil price movements.

According to the thesis this method represents a new approach which discards time-series information into HMM-discretized inputs for Belief Networks thus enhancing the innovative aspects of the research (Page 71).

Explanation of HMM Code from the Thesis

The thesis adopts hmms Python library to execute HMMs for detecting changes in model states. Implementation details with code snippets appear in two sections of the document (Pages 67-69) including the testing phase (Section 4.3.1). We will study the main block of code while detailing its workings for regime detection.

Python Codes From the Thesis:

```
discrete_test = pd.DataFrame(index = test_data[1:].index)
for series_id in datasets:
    path = "./hmms/" + series_id.replace(".", "_") + ".npz"
    if series_id == 'forecast':
        dhmm = hmms.DtHMM.from_file('./hmms/WTISPLC.npz')
    else:
        dhmm = hmms.DtHMM.from_file(path)
    data_diff = test_data[series_id].diff()[1:]
    emit_seq = np.array(data_diff.apply(lambda x: 1 if x > 0 else 0).values)
    (log_prob, s_seq) = dhmm.viterbi(emit_seq)
    discrete_test[series_id] = s_seq
discrete_test.to_csv("./data/test_data.csv")
```

Code Explanation:

1. **Initialization:** The new pandas DataFrame named discrete_test receives values from state discretization. The index of the new pandas DataFrame matches test dataset parameters yet omits its first entry because differencing decreases dataset length by one entry. The loop processes each dataset (series_id) that contains crude oil price (WTISPLC) and macroeconomic indicator values.
2. **Loading HMM:** Each dataset loads its pre-trained HMM model from the stored files located within the ./hmms/ directory. The file path creation system works through dot replacement with underscores in the series_id (the WTISPLC.npz file corresponds to the forecast variable).

   The method hmms.DtHMM.from_file enables users to load discrete-time HMMs (DtHMMs) stored in specified files. The thesis does not indicate its training approach for these HMMs but it suggests hmms library usage to train the models on the same training dataset.

3. **Differencing Data:** A difference of the current series_id time-series data is calculated

from test_data[series_id].diff()[1:]. The differencing process introduces NaN as the first element so the [1:] slice function eliminates it.

4. **Emission Sequence Creation:** A binary emission sequence derived from differenced data is generated through lambda function x: 1 if x > 0 else 0. The model assigns value 1 to price increase datasets while setting all other price changes to value 0.

   The library converts the result into a NumPy array (emit_seq) for processing with hmms.

5. **Viterbi Algorithm**: Due to the dhmm.viterbi(emit_seq) method the Viterbi algorithm finds the most probable hidden state sequence (s_seq) based on the emission sequence. The executed algorithm produces both the log probability of the state sequence (log_prob) and the sequence of states (s_seq) within a returned tuple.

   The detected market conditions appear as values in s_seq because the hidden states show the detected regimes (0, 1, or 2).

6. **Strong Result:** The state sequence (s_seq) receives assignment to its matching column in discrete_test during the current series_id processing.

   The discrete_test DataFrame gets saved as ./data/test_data.csv using CSV format for later use in Bayesian Network prediction.

How the hmm Library operates

The thesis references hmms library (page 29 section 2.2.2) that provides Python library functionality for Hidden Markov Models. This Python library operates with discrete-time HMMs (DtHMM) because of their compatibility with binary emission sequences studied in the thesis. The key relevant features of hmms library for the code implementation involve:

A DtHMM object contains all required parameters of an HMM model structure.

The transition matrix determines the probabilities which govern state transition movements in the HMM structure.The emission matrix contains information about the chance to detect emission values (0 or 1) based on a hidden state. The distribution of initial hidden states contains the probabilities for beginning each hidden state inside the system. The viterbi method applies to analyze hidden state sequences because it determines the most probable hidden state ordering from provided emission sequences to detect regimes.

The from_file method permits importing pre-trained HMM models from files to use trained models with different datasets without manual rebuilds.

The thesis considers HMM training to have been completed before the testing phase based on similar training procedures that were used for the training dataset. The training process would determine transition and emission probabilities from historical data through implementation of the Baum-Welch algorithm (a standard method for HMM training).

INTEGRATION WITH BAYESIAN NETWORK

The HMM-generated discretized states provide input data to the Bayesian Network for oil price projection. The code on Page 68 within In [36] executes a procedure to load discretized test data before removing the WTISPLC forecast column for Bayesian prediction of the forecast variable.

Python Code Snippets

```
discrete_test = pd.read_csv("./data/test_data.csv", index_col=0)
test_real = discrete_test['WTISPLC'].as_matrix()
test_data_new = discrete_test.drop('forecast', axis=1)
testjunction = model.predict(test_data_new)
pred_value_test = testjunction['forecast'].as_matrix()
```

The evaluation method compares model predictions with actual states to determine the 42.86% error rate on the test set (Page 69). The detected error rate demonstrates how well the model predicts the divided market sections before financial decisions can be made.

CHALLENGES AND CONSIDERATIONS

Model Training: The thesis fails to explain the training process of HMMs thus affecting its reproducibility potential. Based on the choice of hidden states and training data quality both play essential roles in achieving effective regime detection.

Binary Encoding: A simple 1/0 binary encoding method might reduce data complexity so it could fail to capture subtle patterns in market trends. Using complex emission model designs would enhance accuracy levels.

Error Rates: The reported error rates reaching 67.86% on validation data and 42.86% on testing data indicate that the crude oil market complexity remains beyond what either the discretization method or Bayesian model can effectively explain. According to the paper (Page 67), the Hill Climbing run should be performed again on the Bayesian Network structure when the error rate exceeds certain thresholds.

Scalability: The procedure works with only a small number of datasets. Future research plans on hundreds of variables would need more efficient structure learning algorithms because of the proposed scope expansion (Page 72).

CONCLUSION

The thesis applies HMMs as a fundamental method for detecting regimes in its crude oil price forecasting system. Time-series data discretization for the model transforms complex oil market dynamics into an applicable form for Bayesian Networks. The hmms library delivers powerful

HMM tools for implementation which especially utilizes Viterbi to determine state sequences. The code applies a transformation to turn real-time price information into distinct groups before using this data to create future price predictions for trading strategy development. The training methods for HMMs exhibit problems because they generate unacceptably high mistakes and provide minimal details about the training process.

Student C writes up to 5 pages that discuss how the network is trained using pgmpy and how the parameters are tested and validated.

The research by Danish A. Alvi "Application of Probabilistic Graphical Models in Forecasting Crude Oil Price" implements Probabilistic Graphical Models (PGMs) including Bayesian Networks to predict crude oil prices through Python programming with pgmpy library. PGMPy Python library enables users to develop and train these networks because it allows them to simulate complex macroeconomic and physical market system dynamics which influence oil price behavior. The discussion explains pgmpy training methods for Bayesian Networks and their parameter evaluation techniques as well as performance validation standards according to the thesis. The main attention is directed towards the document's approach methodology as well as its programming execution and evaluation procedures.

BAYESIAN NETWORK TRAINING WITH pgmpy

The two main requirements in training a Bayesian Network consist of structure learning which identifies the directed acyclic graph that describes variable dependencies and parameter learning which determines conditional probability distributions for each node according to its parent variables. Structure learning and parameter learning tasks are simplified through pgmpy integration which enables the model to extract causal information from available data.

The training process receives its description in Section 3.4 ("Constructing structure of the Crude Oil Markets," Page 43) before actual implementation occurs in Chapter 4 ("Implementation," Page 48). Training data consists of HMM-processed time-series data from EIA and FRED sources following discretization (Section 3.3 Page 42).

The pgmpy program uses Hill Climbing Search as an algorithm for structure learning which applies heuristic search methods. The algorithm starts from an initial network and then performs successive adjustments to improve the model score which can use Bayesian Information Criterion or K2 score to find a balance between model accuracy and complexity. Pgmpy offers structure learning features which the thesis uses according to Ankan resources [1, 2, 3] (Page 9, Section 1.2.3).

The process involves:

Input Data: The training dataset contains discrete variables showing WTI crude oil spot price with symbol WTISPLC and market variables including OPEC supply and OECD demand which have three possible states ranging from 0 to 2.

Algorithm Execution: The initial phase of the Hill Climbing Search algorithm utilizes either a blank graph or a randomly selected one before it begins a sequence of edge operations that enhance score performance. The addition of restrictions about forbidden edges helps maintain structures that remain relevant to the domain.

Output: The structure describes a directed acyclic graph which uses nodes as variables and conditional dependencies through edges.

The thesis does not explicitly show the exact code for structure learning but suggests using pgmpy HillClimbSearch class API. A standard implementation of the process follows references from [1] and [46] to show:

Python Code:

```
from pgmpy.estimators import HillClimbSearch
from pgmpy.estimators import BicScore
# Load discretized training data
train_data = pd.read_csv("./data/train_data.csv")
# Initialize Hill Climbing Search
hc = HillClimbSearch(train_data)
# Use BIC score for structure evaluation
best_model = hc.estimate(scoring_method=BicScore(train_data))
```

The code creates a DAG by using BIC score optimization to avoid model patterns that are too complicated while preventing overfitting behavior. The Hill Climbing method might stop at a local maximum according to the thesis so the algorithm needs to be re-executed when disconnected trees appear in the result (Page 68).

Parameter Learning

The learned network structure helps parameter learning to determine Conditional Probability Distributions (CPDs) for each node according to its parent variables. The parameter estimation process within pgmpy likely selects between Maximum Likelihood Estimation (MLE) and Bayesian Estimation according to standard pgmpy usage (referred to in Ankan [2] page 73).

The process involves:

Input: The learned DAG performs operations on both the discretized training data.

Estimation: CPD computation at each node determines its probabilities through counting the training data state frequency combinations. Analysis of WTISPLC requires OPEC_Supply and OECD_Demand as inputs thus the CPD should provide P(WTISPLC | OPEC_Supply, OECD_Demand) for every valid pairing of states.

Output: A collection of CPDs functions as the definition of the network distribution probabilities.

The parameter learning implementation in pgmpy would follow this typical procedure as discovered in the thesis:

Python Codes:

```
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
# Define the model with the learned structure
model = BayesianModel(best_model.edges())
# Fit the model to estimate CPDs
model.fit(train_data, estimator=MaximumLikelihoodEstimator)
```

The paper does not include specific code but it mentions pgmpy as the framework for implementing Bayesian Networks and performing inferential operations (Page 9). An entirely specified Bayesian Network emerges as the final product.

Data Processing:

The data processing step establishes data compatibility for pgmpy just before training starts.

Discretization: A HMM-based approach discretizes continuous time-series data according to Section 3.3 on Page 42 as explained previously (Artifact 1). A DataFrame with distinct states represents the output.

Splitting: The experimental procedure adopts a three-part division of the dataset into training, validation, and testing sections (Page 40 Section 3.2.2). The training set serves purposes for both parameter and structure models before the evaluation occurs using six performance measures on validation and testing sets.

Cleaning: Through the pandas library the analysis handles missing dataset values while ensuring consistent data maintenance (Section 3.2.1 Page 40).

Bayesian Network Testing

The testing procedure of the Bayesian Network requires using the trained model to forecast the discretized states of WTISPLC from the test dataset followed by an assessment against true WTISPLC states. According to Section 4.3.1 ("Testing," Page 68) of the thesis the testing protocol contains two stages. First HMMs perform the test data discretization before Bayesian Network inference.

The key steps are:

Data Preparation: The test dataset receives the same HMM application which the training data employed (Page 68, In [35]). The DataFrame discrete_test includes discrete values for all variables ranging from WTISPLC to other variables.

Inference: After removing the forecast column (WTISPLC) from the test data for unbiased prediction the Bayesian Network utilizes its predict methods from pgmpy to predict states which are missing.

Python Codes:

```
discrete_test = pd.read_csv("./data/test_data.csv", index_col=0)
test_real = discrete_test['WTISPLC'].as_matrix()
test_data_new = discrete_test.drop('forecast', axis=1)
testjunction = model.predict(test_data_new)
pred_value_test = testjunction['forecast'].as_matrix()
```

Calculation Error: A comparison of predicted states against actual states enables the computation of error rates through the determination of mismatched predictions' percentage. The thesis computes its predictions using a shifted comparison model that predicts the next time period.

Python Codes:

```
error = np.mean(test_real != np.roll(pred_value_test, 1))
print("\nError: ")
print(error * 100)
```

The reported error rate for the test set amounted to 42.86% according to Page 69 which shows the model correctly predicted the regime in 57.14% of cases (Page 69).

Trading Simulation:

The thesis extends testing to a trading simulation, where the predicted states are used to generate trading signals (Page 69, In [39]). The signals are; 0: Short position (sell), Long position (buy) and, No action (hold).

The code below simulates trading starting with one barrel of oil:

```
test_price = pd.DataFrame(test_data['WTISPLC'], columns=['WTISPLC'])
test_signal = pd.DataFrame(testjunction, columns=['forecast'])
test_sheet = pd.concat([test_price, test_signal], axis=1, join='inner')
trades = [test_sheet['WTISPLC'].iloc[0]]
position = False  # True for Long, False for Short
for i in range(len(test_sheet) - 1):
    if test_sheet['forecast'].iloc[i+1] == 0:
        trades.append(trades[-1])
    elif test_sheet['forecast'].iloc[i+1] == 2:
        if position == False:
            position = True
            trades.append(trades[-1])
        else:
            trades.append(test_sheet['WTISPLC'].iloc[i+1])
    else:
        if position == False:
            trades.append(trades[-1])
        else:
            trades.append(test_sheet['WTISPLC'].iloc[i+1])
```

The performance visualization includes three components: actual prices, trading strategy data and an EIA prediction (Page 70). The practical worth of the model is evaluated in this simulation by testing its trading potential to produce profit.

Bayesian Network Validation Methodology

The model tuning and generalization assessment occurs through validation testing on a different validation dataset just before final testing commences. The procedure follows testing principles as described on Page 67 of the source material (In [33]).

Data Preparation: Discretization of the validation set data occurs through HMM operation while the forecast data point becomes inaccessible.

Inference: The Bayesian Network serves to deliver predictions for the forecast variable.

```
vald_real = states['WTISPLC'].as_matrix()
vald_data_new = states.drop('forecast', axis=1)
valdjunction = model.predict(vald_data_new)
pred_value_vald = valdjunction['forecast'].as_matrix()
```

Calculation Error: The error analysis calculates the rate through comparison of predicted and actual states.

```
error = np.mean(vald_real != np.roll(pred_value_vald, 1))
print("\nError: ")
print(error * 100)
```

The validation error rating reaches 67.86% according to Page 67 but demonstrates lower performance than the test set results.

Adjusting the Model:

On page 67 the thesis describes the validation step as an adjustment procedure which occurs when error rates exceed acceptable levels. The structure learning process (Hill Climbing) restarts after unsatisfactory validation errors occur to search for a superior network structure.

"The validation step is to adjust the model if the error is too high. In this case, we can start again by learning the Bayesian model via the Hill Climbing method and observe the change in performance." (Page 67)

The cyclical methodology enables the model to receive additional improvement prior to executing final evaluation. The thesis states that the quality assessment of network structure should evaluate its connectedness (Page 68). The investigation continues with another Hill Climbing iteration to reach a superior local maximum when these disconnected trees appear in the structure.

Several stages of improvement occur to prepare the model for its ultimate testing phase. The thesis recommends studying the network structure quality through a connectedness evaluation (Page 68). Whenever a model structure shows disconnected trees through Hill Climbing it indicates poor fit so the process will begin another search toward better local or global maximum solutions.

Stress Testing

The thesis incorporates stress testing as part of its model validation process since it allows testers to simulate economic distress scenarios to verify model reliability (Page 3, Section 3). The paper fails to present detailed information or programming code for conducting stress testing. Based on the available information stress testing requires using the model to perform simulations through historical crisis data that demonstrates market volatility (the 2008 Financial Crisis is specifically cited on Page 7). The document relates to Rebonato and Denev [56] regarding portfolio management under stressful conditions on Page 77.

Challenges and Considerations:

Multiple obstacles exist during the execution of training, testing and validation methods.

High Error Rates: The model's prediction accuracy suffers based on validation error data at 67.86% as well as test error data at 42.86%. This difficulty mirrors either challenging dynamics in the oil market or issues arising from the discretization method.

Local Maxima in Structure Learning: According to the Hill Climbing algorithm users need to perform multiple runs to achieve better performance results (Page 68).

Lack of Stress Testing Details: Specified stress tests absent from the implementation prevent researchers from accurately assessing how well the model holds up during extreme conditions.

Data Limitations: The thesis uses macroeconomic variables that need additional variables with improved algorithms to enhance forecasting accuracy (Page 72).

Conclusion:

The thesis successfully utilizes *pgmpy* to develop a crude oil price prediction Bayesian Network through the inclusion of structure learning with Hill Climbing Search as well as parameter learning with Maximum Likelihood Estimation. Testing results show a 42.86% accuracy on the test dataset through state prediction before validation required additional changes because of the 67.86% error rate on the separate dataset. The practicality of the simulation test stands strong yet the authors provide minimal details about stress testing implementation. The pgmpy tool facilitates automated discovery of sophisticated patterns between variables although its accuracy remains unstable because of excessive errors and unsatisfactory fitting results. Additional future research directions according to the thesis (Page 72) involve investigating max-min hill climbing algorithms and other variables to boost model accuracy and reliability in actual trading platforms.

STEP 6: Regime Detection

Student A writes 1–2 pages explaining the process of transforming the time series.

Overview of the Transformation Process:

Time-series data containing oil price movements alongside macroeconomic indicators need to be assessed for different market regimes (bears and bulls along with neutrality) through regime detection methods. The regimes transform complicated price patterns into basic categorical conditions that function as network-variable inputs. Data transformation includes steps such as cleaning, differencing to reach stationarity followed by binary emission sequence encoding and the application of HMMs for state inference. The Python library hmms functions as an implementation tool for HMMs while the procedure gets applied to datasets obtained from the Energy Information Administration (EIA) and Federal Reserve Economic Data (FRED).

Time Series Transformation Steps:

The start of time-series analysis requires raw data cleaning followed by missing value management. Page 55 demonstrates how the pandas library performs this task through the following code sample.

Python Codes:

```
import pandas as pd
import numpy as np
data = pd.DataFrame()
for series_id in datasets:
    df = pd.read_csv("./data/" + series_id + ".csv")
    df["DATE"] = pd.to_datetime(df["DATE"])
    df = df.set_index("DATE")
    data[series_id] = df[series_id]
data.to_csv("./data/data.csv")
```

This code:

Loads each dataset (e.g., WTISPLC for WTI crude oil spot price) from CSV files. The DATE column receives datetime formatting while the code establishes it as the index. A single DataFrame (data) results from uniting all available datasets through which time-series variables become column entries. The program saves the cleaned data to data.csv. The process ensures proper formatting along with inconsistency removal in particular elements like dates which creates a basis for upcoming data processing operations.

Data Splitting:

The prepared dataset is divided into separate training and validation and testing sets which help with model development as well as evaluation tasks. The following lines of code illustrate this concept on Page 57:

Codes:

```
train_data = data[:int(0.7 * len(data))]
vald_data = data[int(0.7 * len(data)):int(0.85 * len(data))]
```

```
test_data = data[int(0.85 * len(data)):]
train_data.to_csv("./data/train_data.csv")
vald_data.to_csv("./data/vald_data.csv")
test_data.to_csv("./data/test_data.csv")
```

A division exists for the data into the following parts:

Training Set: The Baysian Network and HMM received training data constituting 70% of the total information.

Validation Set: Performance tuning of the model alongside final evaluation occurs through 15% of data selectable for the purpose.

Testing Set: Final assessment depends on 15% of the total dataset set aside for this purpose.

A proper split of the data allows the model to learn historical information before it uses this knowledge to predict untested data thus delivering forecasting results that mirror actual market conditions.

Differencing and Binary Encoding

The time-series data requires two steps for preparation using HMM-based regime detection: first it becomes stationary through differencing then it gets converted into binary emission sequences. The training set process appears on Page 59 through this code illustration:

Codes:

```
train_diff = pd.DataFrame(index=train_data[1:].index)
for series_id in datasets:
    train_diff[series_id] = train_data[series_id].diff()[1:]
    train_diff[series_id] = train_diff[series_id].apply(lambda x: 1 if x > 0 else 0)
train_diff.to_csv("./data/train_diff.csv")
```

The steps are:

Differencing: Application of diff() method establishes time lag differences between two adjacent periods (price_t - price_{t-1}) to eradicate trends and generate stationary sequences. The complete first row gets eliminated from analysis because differencing generates invalid value NaN for its first element in the series.

Binary Encoding: The difference values get transformed into a binary pattern which sets 1 for rises in price and 0 for price drops or stable conditions. The conversion from differences to binary values occurs through a lambda function which evaluates elements as 1 when x > 0 otherwise returning 0.

Output: The authors stored binary sequences in train_diff.csv file after converting them to a DataFrame (train_diff).

By altering the data to binary encoding the information becomes applicable for HMM processing where hidden states produce sequence emissions.

HMM Application for Regime Detection

The binary emission sequences help us use HMMs to determine market regime sequences. The research uses hmms library to train and implement Hidden Markov Models. The code on Page 61 shows the process:

Codes:

```
import hmms
train_states = pd.DataFrame(index=train_data[1:].index)
for series_id in datasets:
    emit_seq = np.array(train_diff[series_id].values)
    dhmm = hmms.DtHMM.random(3, 2)
    dhmm.baum_welch(emit_seq, 100)
    (log_prob, s_seq) = dhmm.viterbi(emit_seq)
    train_states[series_id] = s_seq
    path = "./hmms/" + series_id.replace(".", "_") + ".npz"
    dhmm.save_params(path)
train_states.to_csv("./data/train_states.csv")
```

The steps are:

A discrete-time HMM (DtHMM) is initialized with 3 hidden states (representing different regimes, e.g., bullish, bearish, neutral) and 2 emission symbols (0 and 1). The random method creates an HMM with random initial parameters.

Training: The Baum-Welch algorithm (baum_welch) is used to train the HMM on the binary emission sequence (emit_seq), optimizing the transition and emission probabilities over 100 iterations.

Student B writes 1–2 pages explaining how the parameters are learned. The Baum-Welch algorithm can be treated as a black box.

In our exploration of financial time-series modeling using Hidden Markov Models (HMMs), determining the appropriate model parameters is a crucial step. These parameters encompass the initial probabilities of each state (π), the likelihood of transitions between hidden states (A), and the probabilities connecting hidden states to observable outcomes (B).

To estimate these, we employ the Baum-Welch algorithm — a specific application of the broader Expectation-Maximization (EM) framework — though the computational mechanics behind it can be quite intricate.

This algorithm works through an iterative process, progressively adjusting parameter values to enhance the likelihood of the given data under the model. It begins with initial estimates, which may stem from prior domain insights or be set arbitrarily, and through successive refinements, aligns these estimates more closely with the observed sequences.

Within the framework of our study, time-series records such as crude oil prices and various macroeconomic variables serve as inputs. These datasets undergo preprocessing steps like noise reduction and differencing to stabilize their statistical properties. The refined data then inform the Baum-Welch procedure as it determines the HMM parameters that most accurately reflect the underlying patterns in the data.

Baum-Welch Algorithm:

1. **Initialization**:
    - Initialize the initial state probabilities πi for each state i.
    - Initialize the transition probabilities aij for transitioning from state i to state j.
    - Initialize the emission probabilities bj(k) for observing symbol k in state j.
2. **E-step (Expectation Step)**:
    - Compute the forward probabilities using the forward algorithm:

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, q_t = i \mid \lambda)$$

    - Compute the backward probabilities βt(i) using the backward algorithm.
    - Calculate the expected state occupancy probabilities γt(i).
    - Calculate the expected transition probabilities ξt(i,j).
3. **M-step (Maximization Step)**:
    - Update the initial state probabilities πi.

   - Update the transition probabilities aij.
   - Update the emission probabilities bj(k).

4. **Iteration**:
   - Repeat steps 2 and 3 until convergence, which can be determined by checking if the parameters have stabilized or if the likelihood $P(O|\lambda)$ has stopped improving significantly.

The Baum-Welch algorithm is an HMM-specific implementation of the EM algorithm. It leverages the particular structure of HMMs to efficiently estimate their parameters. While EM provides a general framework, Baum-Welch adapts it to the unique requirements of HMMs, incorporating the temporal dynamics and structure inherent in sequence data. This specialization makes Baum-Welch more efficient and effective for HMM parameter estimation compared to applying a generic EM algorithm to HMMs.

Student C writes 1–2 pages on finding the most likely sequence of hidden states as the underlying regimes that generated the sequence of observed emissions. The Viterbi algorithm should be treated as a black box.

Finding the most likely sequence of hidden regimes that explains the observed time series is a crucial step following model estimation in the context of crude oil price forecasting using Hidden Markov Models (HMMs). These hidden states may be associated with bull, bear, or stagnant market circumstances. As we have discussed above about these regimes in step 2.

To do this, we use the Viterbi algorithm, a dynamic programming technique that uses the sequence of observed emissions (such changes in oil prices) to determine the most likely sequence of hidden states.

In our workflow, the Viterbi algorithm functions as a black box, meaning we are not going into the mathematics behind the algorithm we are just going to work on the flow of algorithm. A series of hidden states is created by the Viterbi method after the HMM parameters, such as transition probabilities, emission probabilities, and initial state probabilities, have been determined (for example, using the Baum-Welch algorithm, as in part b of step 6). The regime most likely responsible for the corresponding measured emission at that time step is represented by each hidden state in this sequence.

The time series data and the trained HMM are the inputs for Viterbi technique . A series of hidden states is the result, which can subsequently be mapped back to qualitative interpretations, including classifying a certain time period as a bear, bull, or stagnant market regime (explained in parts of step 2). This classification is essential for financial decision-making because it enables regime-aware tactics that adjust to shifting market dynamics.

We concentrate on the input-output behaviour of the Viterbi algorithm by considering it as a "black box," avoiding an exploration of its internal workings. The main conclusion is that the algorithm maximises the total probability of the sequence under the trained model by identifying the one most likely sequence of hidden regimes over the whole period. In contrast to just determining the most likely state at each moment separately, it takes the entire journey into account collectively, resulting in a more logical regime segmentation.

As a result, the Viterbi method offers a vital link between the practical detection of latent regimes in financial time series and the probabilistic model built on historical data.

Step 7 As a group, the team identifies the latent meaning behind each hidden state.

An unobserved or latent variable that affects the system's observable outputs (or emissions) is referred to as a hidden state in the context of Hidden Markov Models (HMMs). A potential regime or condition that might account for the observed data is represented by each concealed state.

In order to understand what the model's hidden states mean in the real world, the team will examine them in this stage. For instance, each hidden state may represent a distinct market condition if the HMM is used to model crude oil prices, such as:

- Bull Market (Rising Prices): This may be a covert state that denotes times when prices are rising sharply.
- Bear Market (Falling Prices): This could indicate times when prices are dropping significantly.

- Flat prices, or a stagnant market, could indicate times when there is little change in prices.

Investigating the nature of each hidden state in the model and providing insightful real-world interpretations is the work at hand. For example, our team will have to explain what market or economic factors those hidden states might correlate to (such as periods of high volatility, geopolitical tensions, or economic booms) if the model identifies various market conditions based on price movements.

Step 8 For the Hill working example, each person carefully re-reads Section 4.2.6 of the paper.

Hill climbing is a structure learning algorithm that is based on scores. The algorithm begins with a basic structure, which could be arbitrary or empty. Suggests making local adjustments by adding, removing, or reversing edges. Determines if every modification enhances a scoring function (such as K2 or BIC), only retains the modification if it raises the score. Continues until there is no more progress (i.e., it is stuck on a local maximum).

Step 9

a. implements the minimal working example of a Hill Climb search, and

please find in the attached python notebook

b. identifies the latent meaning behind each hidden state.

Each hidden state from the HMM shows some signification related to the data and how they interact with different independent variables of input. As we describe our hidden state above as bull, bear and stagnant regimes, they play a very important role in describing the dependent variables via the main features. We will try to find these connections in the next and final GWP where we combine all the data and implemented algorithms together to see the result and conclude our results.

Note: In the attached python notebook we have shared a very simple example of how pgmpy modules works and this is how we will do in next GWP to find the DAG for our dataset.

REFERENCES:

1. Alvi, Danish A. Application of Probabilistic Graphical Models in Forecasting Crude Oil Price. 2018. University College London, Dissertation. https://arxiv.org/abs/1804.10869

2. Murphy, K. P. (2022). Probabilistic Machine Learning: An Introduction. MIT Press