Social Upheaval Composite Index: Mathematical Framework

Component Dimensions

A. Political Violence & Instability (25% weight)

Rationale: Direct threats to social order create immediate cultural anxiety

Measurable Indicators:

- Political assassinations (presidents, major politicians, activists)
- Domestic terrorist attacks with political motivation
- Major riots/civil unrest (number and severity)
- Political protests (frequency and size)
- Government instability (resignations, impeachments)

Mathematical Formula:

$$PV_d = \min\left(20A_d + 15T_d + 10R_d + 5P_d + 8G_d, 100\right)$$

Where:

- PV_d = Political Violence score for decade d
- A_d = Number of major political assassinations in decade d
- T_d = Number of domestic terrorist attacks in decade d
- R_d = Number of major riots/civil unrest events in decade d
- P_d = Number of large-scale political protests in decade d
- G_d = Number of government crises (resignations, impeachments) in decade d

Code Implementation:

```
def calculate_political_violence_score(decade_data):
    score = 0

    # High-impact events (weighted heavily)
    score += decade_data['assassinations_major'] * 20
    score += decade_data['terrorist_attacks_domestic'] * 15
    score += decade_data['riots_major'] * 10

    # Medium-impact events
    score += decade_data['protests_large'] * 5
    score += decade_data['government_crises'] * 8

    # Cap at 100, normalize
    return min(score, 100)
```

B. Institutional Trust Erosion (20% weight)

Rationale: Loss of faith in institutions creates societal paranoia themes

Measurable Indicators:

- Major political scandals (Watergate-level)
- Supreme Court controversial decisions
- Military/intelligence failures or scandals
- Media credibility crises
- Electoral integrity questions

Mathematical Formula:

$$IT_d = \min\left(25S_d + 10C_d + 15I_d + 12E_d, 100\right)$$

Where:

- IT_d = Institutional Trust score for decade d
- S_d = Number of major political scandals in decade d

- C_d = Number of controversial Supreme Court decisions in decade d
- I_d = Number of intelligence/military scandals in decade d
- E_d = Number of electoral controversies in decade d

Code Implementation:

```
def calculate_institutional_trust_score(decade_data):
    score = 0

    # Major scandals with lasting impact
    score += decade_data['major_scandals'] * 25
    score += decade_data['supreme_court_controversial'] * 10
    score += decade_data['intelligence_scandals'] * 15
    score += decade_data['electoral_controversies'] * 12

    return min(score, 100)
```

C. Economic Stress & Inequality (15% weight)

Rationale: Economic anxiety drives demand for films exploring systemic problems

Measurable Indicators:

- Recession severity and duration
- Unemployment peaks
- Income inequality measures (Gini coefficient changes)
- Major corporate/financial scandals
- Housing/cost of living crises

Mathematical Formula:

$$ES_d = \min\left(\min\left(M_d \cdot U_d, 40\right) + \max\left(\Delta G_d \cdot 30, 0\right) + 15F_d, 100\right)$$

Where:

- ES_d = Economic Stress score for decade d

- M_d = Number of months in recession during decade d
- U_d = Peak unemployment rate in decade d

- δG_d = Change in Gini coefficient during decade d (only positive changes count)
- F_d = Number of major financial scandals in decade d

Code Implementation:

```
def calculate_economic_stress_score(decade_data):
    score = 0

    # Recession impact
    recession_severity = decade_data['recession_months'] * decade_data['une
mployment_peak']
    score += min(recession_severity, 40)

    # Inequality changes
    gini_change = decade_data['gini_coefficient_change'] * 100
    score += max(gini_change, 0) * 30  # Only increases count

    # Financial scandals
    score += decade_data['major_financial_scandals'] * 15

    return min(score, 100)
```

D. External Threats & Conflicts (20% weight)

Rationale: External dangers create paranoid/thriller cultural themes

Measurable Indicators:

- War involvement (duration, casualties, controversy)
- International terrorist threats
- Cold War tensions/nuclear fears

- Foreign interference in elections
- Pandemic/health crises

Mathematical Formula:

$$ET_d = \min\left(\min\left(W_d \cdot C_d \cdot V_d, 35\right) + 10T_d + 15I_d + 20P_d, 100\right)$$

Where:

- ET_d = External Threats score for decade d
- W_d = Number of years at war during decade d
- C_d = Casualty rate (deaths per 1000 troops per year)
- V_d = War controversy factor (1 = popular, 5 = highly controversial)
- T_d = Terror threat level (1-10 scale)
- I_d = Foreign interference incidents in decade d
- P_d = Pandemic severity score (0-5 scale based on deaths/disruption)

Code Implementation:

```
def calculate_external_threats_score(decade_data):
    score = 0

    # War involvement
    war_impact = (decade_data['war_years'] * decade_data['casualty_rate'] *
                  decade_data['controversy_factor'])
    score += min(war_impact, 35)

    # Terrorism/security threats
    score += decade_data['terror_threat_level'] * 10
    score += decade_data['foreign_interference'] * 15

    # Health/pandemic crises
    score += decade_data['pandemic_severity'] * 20
```

```
    return min(score, 100)
```

E. Social Fragmentation (20% weight)

Rationale: Division and polarization drive demand for films exploring "us vs them" themes

Measurable Indicators:

- Political polarization measures
- Racial/ethnic tensions and incidents
- Generational conflicts
- Regional divisions
- Information/media fragmentation

Mathematical Formula:

$$SF_d = \min\left(25P_d + 15R_d + 20M_d + 15D_d + 25C_d, 100\right)$$

Where:

- SF_d = Social Fragmentation score for decade d
- P_d = Political polarization index for decade d (0-4 scale)
- R_d = Number of major racial tension incidents in decade d
- M_d = Media fragmentation index for decade d (0-5 scale)
- D_d = Disinformation prevalence score for decade d (0-4 scale)
- C_d = Cultural conflict intensity score for decade d (0-4 scale)

Code Implementation:

```
def calculate_social_fragmentation_score(decade_data):
    score = 0

    # Polarization metrics
```

```
    score += decade_data['political_polarization_index'] * 25
    score += decade_data['racial_tension_incidents'] * 15

    # Information environment
    score += decade_data['media_fragmentation_index'] * 20
    score += decade_data['disinformation_prevalence'] * 15

    # Regional/cultural divisions
    score += decade_data['cultural_conflict_intensity'] * 25

    return min(score, 100)
```

Master Composite Index

Mathematical Formula:

$$CUI_d = w_{pv} \cdot PV_d + w_{it} \cdot IT_d + w_{es} \cdot ES_d + w_{et} \cdot ET_d + w_{sf} \cdot SF_d$$

Where:

- CUI_d = Composite Upheaval Index for decade d
- w_{pv} = 0.25 (Political Violence weight)
- w_{it} = 0.20 (Institutional Trust weight)
- w_{es}= 0.15 (Economic Stress weight)
- w_{et} = 0.20 (External Threats weight)
- w_{sf} = 0.20 (Social Fragmentation weight)

Constraint: $\sum w_i = 1.0$

Weighted Aggregation

```
def calculate_composite_upheaval_index(decade_data, weights=None):

    if weights is None:
        weights = {
```

```
        'political_violence': 0.25,
        'institutional_trust': 0.20,
        'economic_stress': 0.15,
        'external_threats': 0.20,
        'social_fragmentation': 0.20
    }

# Calculate component scores
pv_score = calculate_political_violence_score(decade_data)
it_score = calculate_institutional_trust_score(decade_data)
es_score = calculate_economic_stress_score(decade_data)
et_score = calculate_external_threats_score(decade_data)
sf_score = calculate_social_fragmentation_score(decade_data)

# Weighted composite
composite_score = (
    pv_score * weights['political_violence'] +
    it_score * weights['institutional_trust'] +
    es_score * weights['economic_stress'] +
    et_score * weights['external_threats'] +
    sf_score * weights['social_fragmentation']
)

return {
    'composite_score': composite_score,
    'components': {
        'political_violence': pv_score,
        'institutional_trust': it_score,
        'economic_stress': es_score,
        'external_threats': et_score,
        'social_fragmentation': sf_score
    }
}
```

Alternative Aggregation Methods

1. Multiplicative Model (Crisis Amplification)

Mathematical Formula:

$$CUI_{mult,d} = \min\left(\frac{\sum_{i=1}^{5} C_{i,d}}{5} \cdot \alpha_d, 100\right)$$

Where:

$$\alpha_d = \begin{cases} 1.5 & \text{if } \sum_{i=1}^{5} \mathbf{1}_{C_{i,d}>70} \geq 3 \\ 1.2 & \text{if } \sum_{i=1}^{5} \mathbf{1}_{C_{i,d}>70} = 2 \\ 1.0 & \text{otherwise} \end{cases}$$

- $C_{i,d}$ = Component score i for decade d
- $1Ci, d > 70$ = Indicator function (1 if component > 70, 0 otherwise)
- α_d = Amplification factor based on number of high-scoring components

```
# Assumes components amplify each other during true upheaval
def multiplicative_upheaval_index(components):
    base_score = sum(components.values()) / len(components)
    amplification = 1.0

    # If multiple components are high, amplify the effect
    high_components = sum(1 for score in components.values() if score > 70)
    if high_components >= 3:
        amplification = 1.5
    elif high_components >= 2:
        amplification = 1.2

    return min(base_score * amplification, 100)
```

2. Peak-Sensitive Model

Mathematical Formula:

$$CUI_{peak,d} = 0.7 \cdot C_{max,d} + 0.3 \cdot C_{avg,d}$$

```
# Emphasizes the highest single component (worst crisis dominates)
def peak_sensitive_upheaval_index(components):
    max_component = max(components.values())
    avg_component = sum(components.values()) / len(components)

    # Weight toward the peak crisis, but include overall level
    return (max_component * 0.7) + (avg_component * 0.3)
```

Validation Framework

Historical Validation Tests

Rank Correlation Formula:

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)}$$

Where:

- ρ = Spearman's rank correlation coefficient
- d_i = Difference between actual rank and expected rank for decade i
- n = Number of decades being compared

Expected vs Actual Ranking Test:

$$Validation_{score} = \begin{cases} \text{Strong} & \text{if } \rho > 0.7 \\ \text{Moderate} & \text{if } 0.4 < \rho \leq 0.7 \\ \text{Weak} & \text{if } \rho \leq 0.4 \end{cases}$$

```
def validate_index_against_history(decades_data):
    results = []
```

```
    for decade, data in decades_data.items():
        score = calculate_composite_upheaval_index(data)

        results.append({
            'decade': decade,
            'composite_score': score['composite_score'],
            'expected_rank': get_historical_expectation(decade),
            'actual_rank': None  # To be calculated
        })

    # Rank decades by composite score
    results.sort(key=lambda x: x['composite_score'], reverse=True)
    for i, result in enumerate(results):
        result['actual_rank'] = i + 1

    return results

def get_historical_expectation(decade):
    # Expert/historical consensus on most turbulent decades
    rankings = {
        1960: 1,  # Assassinations, Vietnam, civil rights
        1970: 2,  # Watergate, oil crisis, Vietnam end
        2020: 3,  # COVID, Jan 6, polarization
        1940: 4,  # WWII
        2000: 5,  # 9/11, Iraq War
        1930: 6,  # Depression
        # ... etc
    }
    return rankings.get(decade, 10)
```

Sensitivity Analysis

Weight Optimization Formula:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \left| \text{corr}\left(\sum_{i=1}^{5} w_i \cdot C_{i,d}, T_d \right) \right|$$

Subject to:

$$\sum_{i=1}^{5} w_i = 1 \tag{1}$$

$$w_i \geq 0 \quad \forall i \tag{2}$$

$$0.05 \leq w_i \leq 0.50 \quad \forall i \tag{3}$$

Where:

- w^* = Optimal weight vector
- T_d = Number of political thriller films in decade d
- $corr(\cdot, \cdot)$ = Pearson correlation coefficient
- Constraints ensure all components contribute meaningfully (5%-50% range)

```
def test_weight_sensitivity(decades_data):
    # Test different weighting schemes
    weight_schemes = [
        {'political_violence': 0.4, 'institutional_trust': 0.15, ...},  # Violence-heavy
        {'political_violence': 0.1, 'institutional_trust': 0.4, ...},   # Institution-heavy
        # Equal weights, etc.
    ]

    correlations = []
    for weights in weight_schemes:
        scores = [calculate_composite_upheaval_index(data, weights)
                  for data in decades_data.values()]
        correlation_with_thrillers = calculate_correlation(scores, thriller_counts)
        correlations.append(correlation_with_thrillers)

    return correlations
```