A Machine Learning Model for Turning Point Prediction in the Oil Market

CISC-5800-L01 Machine Learning

Fangyuan Xu

Background

- **Crude Oil** is a primary energy source with a big impact on the global economy.
- The oil price is determined by **Supply and Demand** in the physical market
- Accurately identifying oil price **Turning Points** is essential economic planning and risk management.

Objective

Implement a machine learning model to **predict turning points in oil supply chain dynamics** by integrating multi-source data

- supply data
- Inventory data
- policy events
- oil price data

The model aims to detect market shifts early

and support timely decision-making.

Data

- supply and demand

JODI oil supply data （https://data.nasdaq.com/databases/JODI ）

EIA（Energy Information Administration）-*Commercial* crude oil inventory(WCESTUS1)：

（https://api.eia.gov/v2/petroleum/stoc/wstk/data/）

- policy events

OPEC Events Data （https://www.opec.org ）

- oil price data

WTI Crude Oil Prices （https://fred.stlouisfed.org/series/DCOILWTICO ）

Final Integrated Dataset

Data Dictionary

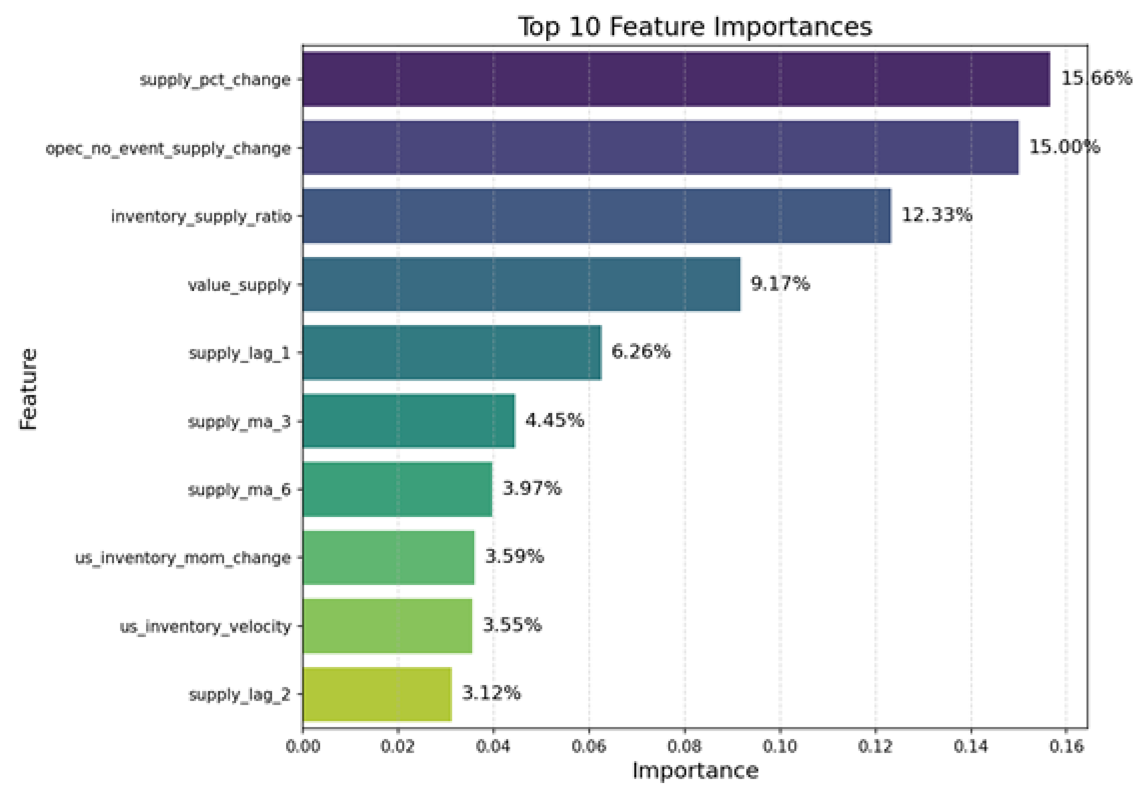
Variable Name	Expected Data Type	Description
value_supply	float	Monthly oil supply (thousand barrels per day)
value_wti	float	WTI crude oil price (USD per barrel)
is_turning_point	int (binary)	Supply turning point indicator (1 = turning point, 0 = no)
supply_pct_change	float	Month-over-month supply change rate (%)
price_pct_change	float	Month-over-month price change rate (%)
supply_lag_1	float	1-month lagged supply value
supply_lag_2	float	2-month lagged supply value
supply_lag_3	float	3-month lagged supply value
supply_ma_3	float	3-month moving average of supply
supply_ma_6	float	6-month moving average of supply
us_inventory_level	float	US crude oil inventory level (million barrels)
us_inventory_mom_change	float	US inventory month-over-month change
us_inventory_5y_percentile	float	US inventory 5-year historical percentile
us_inventory_velocity	float	US inventory turnover rate (inventory/supply)
us_inventory_acceleration	float	US inventory change acceleration
opec_no_event_supply_change	float	Supply change during OPEC non-event periods
inventory_supply_ratio	float	Inventory to supply ratio
date	datetime	Data date (monthly)
country	string	Country code
energy	string	Energy type (fixed as "OIL")

Total number of key columns: 20

Time Frequency

- All data aligned to **monthly** frequency
- Month-end values used for consistency
- Time range: **2002-2024**

Feature Importance



1.Supply Data

(Total:59%%)

2.Policy Data

(Total: 31%)

3.Inventory Data

(Total: 1%)

Final Model 4

> models > versions > v4_final.py > feature_engineering

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.ensemble import RandomForestClassifier
import os
import joblib
from sklearn.metrics import classification_report, confusion_matrix

def feature_engineering(df):
    """
    Perform feature engineering on the dataset
    """
    # Handle categorical variables
    df = df.copy()

    # Ensure is_turning_point is binary
    df['is_turning_point'] = df['is_turning_point'].astype(int)

    # Drop unnecessary columns
    columns_to_drop = ['energy', 'code', 'country', 'date', 'notes', 'event_type']
    df = df.drop(columns=columns_to_drop)

    # Replace infinite values with NaN
    df = df.replace([np.inf, -np.inf], np.nan)

    # Fill NaN values with 0
    df = df.fillna(0)

    # Ensure all features are numeric
    numeric_columns = df.select_dtypes(include=[np.number]).columns
    df = df[numeric_columns]
```

```python
def feature_engineering(df):
    # Standardize features (except target variable)
    features_to_scale = [col for col in df.columns if col != 'is_turning_point']
    scaler = StandardScaler()
    df[features_to_scale] = scaler.fit_transform(df[features_to_scale])

    return df

def calculate_rsi(series, window):
    """Calculate Relative Strength Index"""
    delta = series.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    return 100 - (100 / (1 + rs))

def train_model(df):
    """
    Train the model using the preprocessed features
    """
    # Prepare features and target variable
    X = df.copy()
    y = X.pop('is_turning_point')  # Use turning point as prediction target

    # Split training and test sets
    train_size = int(len(X) * 0.8)
    X_train = X[:train_size]
    X_test = X[train_size:]
    y_train = y[:train_size]
    y_test = y[train_size:]

    # Train model
    model = RandomForestClassifier(
        n_estimators=200,
        max_depth=10,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42
    )
```

80/20 train-test split

```python
def train_model(df):
    # Train model
    model = RandomForestClassifier(
        n_estimators=200,
        max_depth=10,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42
    )
    model.fit(X_train, y_train)

    # Evaluate model
    y_pred = model.predict(X_test)

    # Print classification report
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    # Print confusion matrix
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    # Feature importance
    feature_importance = pd.DataFrame({
        'feature': X.columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=False)

    # Plot feature importance
    plt.figure(figsize=(10, 6))
    sns.barplot(x='importance', y='feature', data=feature_importance)
    plt.title('Feature Importance')
    plt.tight_layout()
    plt.savefig('data/plots/feature_importance.png')
    plt.close()

    return model, feature_importance
```

```python
def train_model(df):
    plt.close()

    return model, feature_importance

def main():
    """
    Main function to run the analysis
    """
    # Load data
    print("Loading data...")
    data = pd.read_csv('data/processed/enhanced_data_v3.csv')

    # Feature engineering
    print("\nPerforming feature engineering...")
    df_features = feature_engineering(data)

    # Train model
    print("\nTraining model...")
    model, feature_importance = train_model(df_features)

    # Save results
    os.makedirs('data/models', exist_ok=True)
    joblib.dump(model, 'data/models/supply_chain_model.joblib')
    feature_importance.to_csv('data/processed/feature_importance.csv', index=False)

    print("\nAnalysis complete!")
    print("- Model saved to: data/models/supply_chain_model.joblib")
    print("- Feature importance saved to: data/processed/feature_importance.csv")

if __name__ == "__main__":
    main()
```

Definition of Turning Point

- A point where oil supply changes direction
- Can be either a peak (highest point) or a valley (lowest point)
- Marked as 1 (turning point) or 0 (not a turning point)

How to Find Turning Points

RandomForestClassifier

- An ensemble learning method using multiple decision trees
- Good for handling non-linear relationships and feature interactions

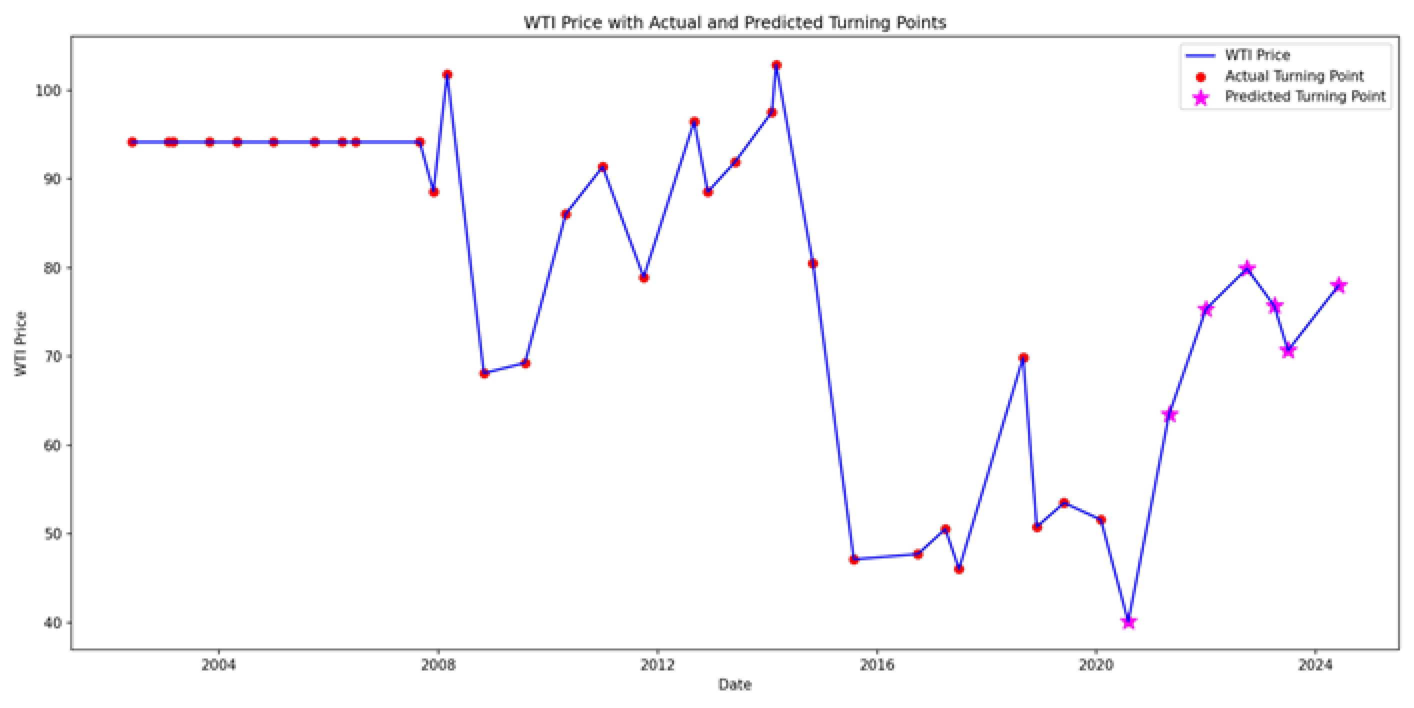
Convert `is_turning_point` to binary integer type

```
def mark_turning_points(data, value_col='value_supply', window=3):
    values = data[value_col].values
    is_turning = np.zeros(len(values), dtype=int)
    half_w = window // 2

    for i in range(half_w, len(values) - half_w):
        window_slice = values[i - half_w: i + half_w + 1]
        center = window_slice[half_w]
        if center == np.max(window_slice) or center == np.min(window_slice):
            is_turning[i] = 1

    data['is_turning_point'] = is_turning
    return data
```

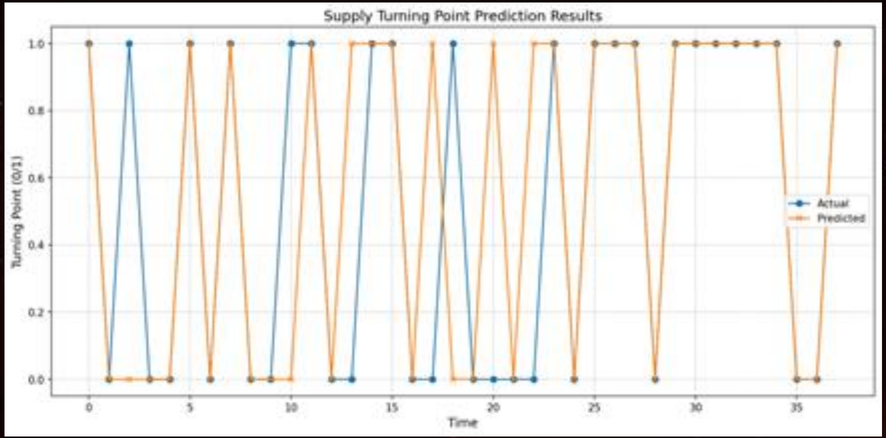
Results

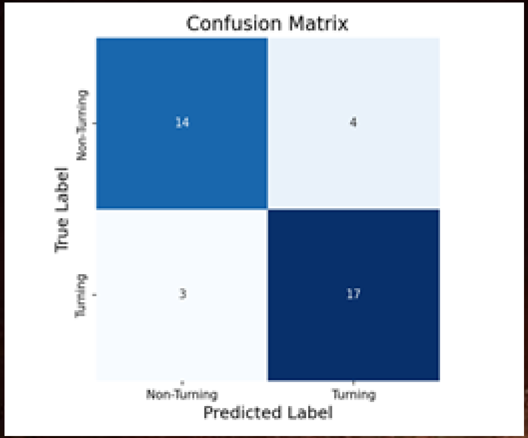


80/20 train-test split

accuracy	0.82	38

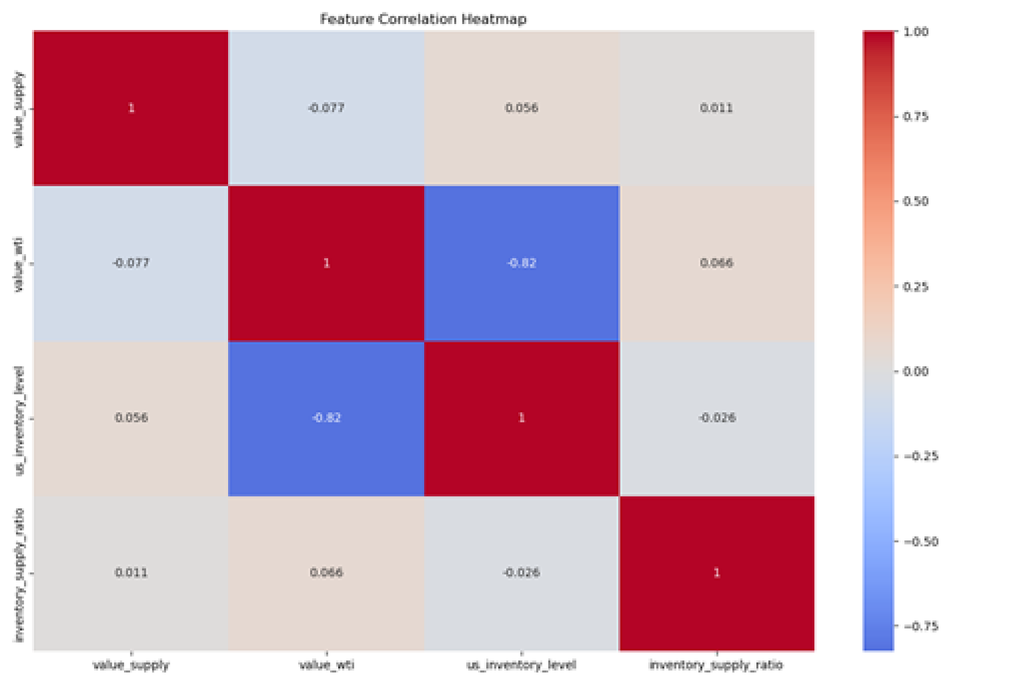
Results





Classification Report:

	precision	recall	f1-score	support
0	0.82	0.78	0.80	18
1	0.81	0.85	0.83	20
accuracy		**0.82**		**38**
macro avg	0.82	0.81	0.81	38
weighted avg	0.82	0.82	0.82	38



The strong negative correlation between US crude oil inventory levels and WTI price is the most noteworthy feature relationship.

Model 1

s > models > versions > v1_baseline.py > feature_engineering

```python
def build_demand_forecast_model(data):
    plt.close()

    return model, scaler, feature_importance

def feature_engineering(data, n_lags=3, ma_windows=[3, 6]):

    for i in range(1, n_lags + 1):
        data[f'supply_lag_{i}'] = data['value_supply'].shift(i)
        data[f'price_lag_{i}'] = data['value_wti'].shift(i)

    data['supply_pct_change'] = data['value_supply'].pct_change()
    data['price_pct_change'] = data['value_wti'].pct_change()
    ⌘L to chat, ⌘K to generate        (parameter) data: Any
    for w in ma_windows:
        data[f'supply_ma_{w}'] = data['value_supply'].rolling(window=w).mean()
        data[f'price_ma_{w}'] = data['value_wti'].rolling(window=w).mean()
    return data

def train_turning_point_classifier(data):
    """Train a classifier to predict supply turning points, now including OPEC event
    # Original features
    base_features = [
        'supply_lag_1', 'supply_lag_2', 'supply_lag_3',
        'price_lag_1', 'price_lag_2', 'price_lag_3',
        'supply_pct_change', 'price_pct_change',
        'supply_ma_3', 'supply_ma_6',
        'price_ma_3', 'price_ma_6'
    ]

    # Add OPEC event features (one-hot encoded)
    opec_features = [col for col in data.columns if col.startswith('opec_')]
    feature_cols = base_features + opec_features

    # Clean data
    data_model = data.dropna(subset=feature_cols + ['is_turning_point'])
    X = data_model[feature_cols]
    y = data_model['is_turning_point']
```

it **defines basic features**, then performs data cleaning and preparation, followed by an 80/20 train-test split in chronological order, trains the model using RandomForestClassifier,

Model 2

```
s > models > versions > v2_opec.py > feature_engineering
def test_stationarity(data):
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

    return result[1] < 0.05

def feature_engineering(data, inventory_data, n_lags=3, ma_windows=[3, 6]):
    """
    Enhanced feature engineering, integrating inventory data
    """
    # Basic features
    for i in range(1, n_lags + 1):
        data[f'supply_lag_{i}'] = data['value_supply'].shift(i)
        data[f'price_lag_{i}'] = data['value_wti'].shift(i)

    # Rate of change features
    data['supply_pct_change'] = data['value_supply'].pct_change()
    data['price_pct_change'] = data['value_wti'].pct_change()

    # Moving average features
    for w in ma_windows:
        data[f'supply_ma_{w}'] = data['value_supply'].rolling(window=w).mean()
        data[f'price_ma_{w}'] = data['value_wti'].rolling(window=w).mean()

    # Integrate inventory features
    inventory_data['date'] = pd.to_datetime(inventory_data['date'])
    data['date'] = pd.to_datetime(data['date'])

    # Merge inventory data
    data = pd.merge(data, inventory_data, on='date', how='left')

    # Inventory-related features
    data['inventory_pct_change'] = data['us_inventory_level'].pct_change()
    data['inventory_supply_ratio'] = data['us_inventory_level'] / data['value_supply'

    # Inventory trend features
    data['inventory_trend'] = data['us_inventory_level'].diff()
    data['inventory_acceleration'] = data['inventory_trend'].diff()
```

Add inventory-related features

Enhanced feature engineering

Model 3

s > models > versions > v3_inventory.py > ...

```python
def feature_engineering(data, inventory_data, opec_data, n_lags=3, ma_windows=[3, 6]):
    # 2. Inventory features
    inventory_data['date'] = pd.to_datetime(inventory_data['date'])
    data['date'] = pd.to_datetime(data['date'])
    data = pd.merge(data, inventory_data, on='date', how='left')

    # Basic inventory features
    data['inventory_pct_change'] = data['us_inventory_level'].pct_change()
    data['inventory_supply_ratio'] = data['us_inventory_level'] / data['value_supply']
    data['inventory_trend'] = data['us_inventory_level'].diff()
    data['inventory_acceleration'] = data['inventory_trend'].diff()
    data['inventory_price_ratio'] = data['us_inventory_level'] / data['value_wti']

    # 3. OPEC features
    opec_data['date'] = pd.to_datetime(opec_data['date'])
    data = pd.merge(data, opec_data[['date', 'event_type']], on='date', how='left')
    data['event_type'] = data['event_type'].fillna('no_event')

    # Ensure all OPEC event types exist
    event_types = ['meeting', 'cut', 'maintain', 'no_agreement', 'extend', 'increase', 'no_
    for event_type in event_types:
        col_name = f'opec_{event_type}'
        data[col_name] = (data['event_type'] == event_type).astype(int)

        # Create interaction features
        data[f'{col_name}_inventory_change'] = data[col_name] * data['inventory_pct_change'
        data[f'{col_name}_supply_change'] = data[col_name] * data['supply_pct_change']

    return data

def train_turning_point_classifier(data):
    """Optimized turning point classifier"""
    # Feature list
    feature_cols = [
        # 1. Supply features
        'supply_lag_1', 'supply_lag_2', 'supply_lag_3',
        'supply_pct_change', 'supply_ma_3', 'supply_ma_6',

        # 2. Price features
```

Add Opec features

Here, the OPEC policy events are divided into seven categories: meeting, cut, maintain, no_agreement, extend, increase, and no_event.

Enhanced feature engineering

Further improvements

1. Data Granularity Expansion:

Changing the time granularity from monthly to daily or weekly can help capture more detailed market fluctuations and short-term policy impacts, thus improving the model’s timeliness and sensitivity.

2. Policy Feature Enhancement:

Refine the classification and quantification of policy events such as OPEC actions, for example by introducing event intensity, duration, or using NLP techniques to analyze policy texts, so as to more accurately reflect the real impact of policies on the market.

[1] O. Durand-Lasserve and A. Pierru, “Modeling world oil market questions: An economic perspective,” *Energy Policy*, vol. 159, p. 112606, 2021. doi: 10.1016/j.enpol.2021.112606.

[2] F. Cheng, T. Fan, D. Fan, and S. Li, “The prediction of oil price turning points with log-periodic power law and multi-population genetic algorithm,” *Energy Economics*, vol. 72, pp. 341–355, 2018. doi: 10.1016/j.eneco.2018.03.038.