Model Development Phase Template

Date	01 December 2024
Team ID	739791
Project Title	Rice Crop Monitoring-Time Series Analysis
Maximum Marks	10 Marks

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summary and training and validation performance metrics for multiple models, presented through respective screenshots.

Initial Model Training Code (5 marks):

1. **Arima Model**

```python
# Function to evaluate ARIMA model
def evaluate_arima_model(train, test, order):
    model = ARIMA(train, order=order)
    model_fit = model.fit()
    predictions = model_fit.forecast(steps=len(test))
    rmse = sqrt(mean_squared_error(test, predictions))
    return rmse

# Function to optimize ARIMA
def optimize_arima(train, test, p_values, d_values, q_values):
    best_score, best_cfg = float("inf"), None
    for p, d, q in itertools.product(p_values, d_values, q_values):
        try:
            order = (p, d, q)
            rmse = evaluate_arima_model(train, test, order)
            if rmse < best_score:
                best_score, best_cfg = rmse, order
            print(f'ARIMA{order} RMSE={rmse:.4f}')
        except:
            continue
    print(f'Best ARIMA{best_cfg} RMSE={best_score:.4f}')
    return best_cfg
```

2. Sarima Model

```python
# Function to evaluate SARIMA model
def evaluate_sarima_model(train, test, order, seasonal_order):
    model = SARIMAX(train, order=order, seasonal_order=seasonal_order)
    model_fit = model.fit(disp=False)
    predictions = model_fit.forecast(steps=len(test))
    rmse = sqrt(mean_squared_error(test, predictions))
    return rmse

# Function to optimize SARIMA
def optimize_sarima(train, test, p_values, d_values, q_values, P_values, D_values, Q_values, m):
    best_score, best_cfg = float("inf"), None
    for p, d, q, P, D, Q in itertools.product(p_values, d_values, q_values, P_values, D_values, Q_values):
        try:
            order = (p, d, q)
            seasonal_order = (P, D, Q, m)
            rmse = evaluate_sarima_model(train, test, order, seasonal_order)
            if rmse < best_score:
                best_score, best_cfg = rmse, (order, seasonal_order)
            print(f'SARIMA{order}x{seasonal_order} RMSE={rmse:.4f}')
        except:
            continue
    print(f'Best SARIMA{best_cfg} RMSE={best_score:.4f}')
    return best_cfg
```

3. Facebook Prophet Model

```python
import pandas as pd
from prophet import Prophet

df = pd.read_csv('rice production across different countries from 1961 to 2021.csv')

# Replace 'your_date_column' and 'your_value_column' with the actual column names from
new = df.rename(columns={'Year': 'ds', 'Value': 'y'})  # Example: Assuming 'Year' is yo

# Instantiate and fit the Prophet model
model = Prophet(seasonality_mode='multiplicative')
model.fit(new)
```

```python
from sklearn.metrics import mean_absolute_error
# Generate predictions on test data
# Ensure y_pred contains predictions corresponding to the test data in 'new'
y_pred = model.predict(new[['ds']])['yhat'] # Predict on the 'ds' values from 'new'

# Calculate root mean squared error
mae = mean_absolute_error(new['y'].values, y_pred)
print('MAE:', mae)
```

Model Validation and Evaluation Report (5 marks):

<table>
<tr><th>Model</th><th>Summary</th><th>Training and Validation Performance Metrics</th></tr>
<tr><td>Arima Model</td><td>

- Combines three components:

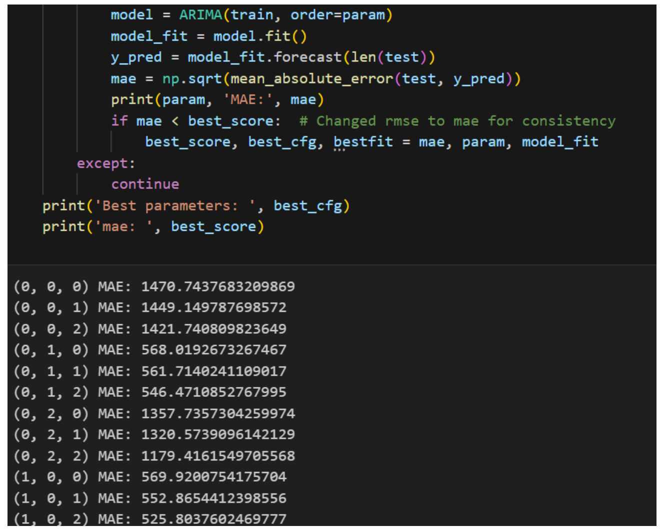
AR (AutoRegression), I (Integration), MA (Moving Average)

Requires the time series to be **stationary** (constant mean and variance over time).

Use Cases:

- Works well for univariate time series with linear trends and seasonality.
- Suitable for short-term forecasting.
-

</td><td>



```
            model = ARIMA(train, order=param)
            model_fit = model.fit()
            y_pred = model_fit.forecast(len(test))
            mae = np.sqrt(mean_absolute_error(test, y_pred))
            print(param, 'MAE:', mae)
            if mae < best_score:  # Changed rmse to mae for consistency
                best_score, best_cfg, bestfit = mae, param, model_fit
        except:
            continue
    print('Best parameters: ', best_cfg)
    print('mae: ', best_score)

(0, 0, 0) MAE: 1470.7437683209869
(0, 0, 1) MAE: 1449.149787698572
(0, 0, 2) MAE: 1421.740809823649
(0, 1, 0) MAE: 568.0192673267467
(0, 1, 1) MAE: 561.7140241109017
(0, 1, 2) MAE: 546.4710852767995
(0, 2, 0) MAE: 1357.7357304259974
(0, 2, 1) MAE: 1320.5739096142129
(0, 2, 2) MAE: 1179.4161549705568
(1, 0, 0) MAE: 569.9200754175704
(1, 0, 1) MAE: 552.8654412398556
(1, 0, 2) MAE: 525.8037602469777
```

</td></tr>
<tr><td>Sarima Model</td><td>

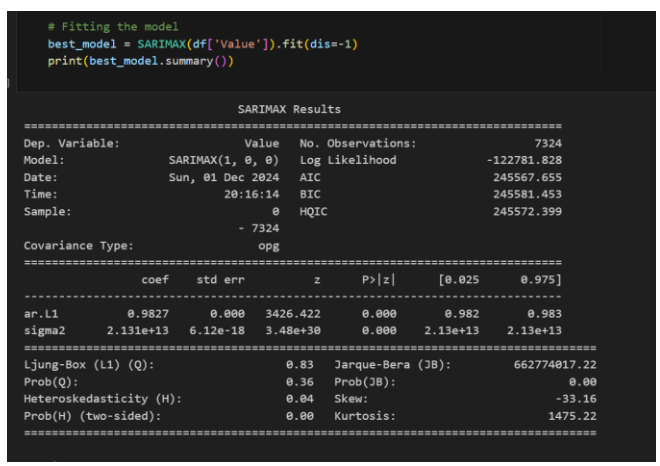
2. SARIMAX (Seasonal AutoRegressive Integrated Moving Average with eXogenous factors)

- Features the Extension of ARIMA

Use Cases:

- Best for time series data with **seasonal patterns** (e.g., monthly sales).
- Incorporates external variables to improve accuracy.

</td><td>



```
    # Fitting the model
    best_model = SARIMAX(df['Value']).fit(dis=-1)
    print(best_model.summary())

                               SARIMAX Results
==============================================================================
Dep. Variable:                  Value   No. Observations:                 7324
Model:               SARIMAX(1, 0, 0)   Log Likelihood             -122781.828
Date:                Sun, 01 Dec 2024   AIC                         245567.655
Time:                        20:16:14   BIC                         245581.453
Sample:                             0   HQIC                        245572.399
                               - 7324
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9827      0.000   3426.422      0.000       0.982       0.983
sigma2      2.131e+13   6.12e-18   3.48e+30      0.000    2.13e+13    2.13e+13
===================================================================================
Ljung-Box (L1) (Q):                   0.83   Jarque-Bera (JB):          662774017.22
Prob(Q):                              0.36   Prob(JB):                          0.00
Heteroskedasticity (H):               0.04   Skew:                            -33.16
Prob(H) (two-sided):                  0.00   Kurtosis:                       1475.22
===================================================================================
```

</td></tr>
</table>

<table>
<tr>
<td>Facebook Prophet model</td>
<td>

Features:

- Developed by Facebook, designed for business forecasting with a focus on **non-statisticians**.
- Automatically handles:

Trends (linear or logistic growth),Seasonality (daily, weekly, yearly), Holidays/events as additional regressors.

Use Cases:

- Robust for time series with irregular or missing data.
- Suitable for datasets with holidays or special events.
- Highly customizable and interpretable.

</td>
<td>

```
# Instantiate and fit the Prophet model
model = Prophet(seasonality_mode='multiplicative')
model.fit(new)
```

```
Importing plotly failed. Interactive plots will not work.
20:18:34 - cmdstanpy - INFO - Chain [1] start processing
20:18:35 - cmdstanpy - INFO - Chain [1] done processing

<prophet.forecaster.Prophet at 0x190b1046d50>
```

</td>
</tr>
</table>