Model Optimization and Tuning Phase Template

Date	01 December 2024
Team ID	Faculty
Project Title	Unemployed Insurance Beneficiary Forecasting
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining time series models for forecasting and peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

HyperParameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters
ARIMA Model 1	We define a hyperparameter space with different combinations of ARIMA orders and seasonal orders. We then perform hyperparameter tuning using GridSearchCV, which evaluates each combination of hyperparameters and returns the best-performing model.

Arima Hyperparameter

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Assuming 'df1' is your DataFrame with columns 'ds' and 'y'
# Split the data into train and test sets
train_size = int(len(df1) * 0.8)
train_data, test_data = df1[:train_size], df1[train_size:]

# Define X_train and y_train for ARIMA
X_train = train_data
```

```
# Define hyperparameter space
param_grid = {
    'order': [(1,1,1), (1,1,2), (2,1,1), (2,1,2), (3,1,1), (3,1,2)],
    'seasonal_order': [(1,1,1,12), (1,1,2,12), (2,1,1,12), (2,1,2,12)]
}

# Define custom scorer for ARIMA model
def arima_scorer(params, X, y):
    order = params['order']
    seasonal_order = params['seasonal_order']
    model = ARIMA(X, order=order, seasonal_order=seasonal_order)
    model_fit = model.fit()
    y_pred = model_fit.forecast(steps=len(y))[0]
    return -mean_squared_error(y, y_pred)

# Perform hyperparameter tuning
grid_search = GridSearchCV(estimator=None, param_grid=param_grid,
                           scoring=arima_scorer, cv=5)
grid_search.fit(X_train)

# Print best hyperparameters and score
print("Best Parameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
```

> **hyperparameters for the ARIMA model are:**
> **Best Parameters: {'order': (2, 1, 2), 'seasonal_order': (1, 1, 1, 12)}**

Performance Metric

```
[59] from sklearn.metrics import mean_squared_error, mean_absolute_error, median_absolute_error, r2_score

[60] mean_squared_error(test['y'], predictions_arima)
55678693.1990747

[ ] mean_absolute_error(test['y'], predictions_arima)
3442.001494456475

[62] median_absolute_error(test['y'], predictions_arima)
932.686511014218

[63] r2_score(test['y'],predictions_arima)
-0.1306393311185059
```

SARIMA

Model 2

We define a hyperparameter space with different combinations of SARIMA orders, seasonal orders, and trends. We then perform hyperparameter tuning using GridSearchCV, which evaluates each combination of hyperparameters and returns the best-performing model.

Sarima Hyperparameter

```
import pandas as pd
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Define hyperparameter space
param_grid = {
    'order': [(1,1,1), (1,1,2), (2,1,1), (2,1,2), (3,1,1), (3,1,2)],
    'seasonal_order': [(1,1,1,12), (1,1,2,12), (2,1,1,12), (2,1,2,12)],
    'trend': ['n', 'c', 't', 'ct']
}

# Define custom scorer for SARIMA model
def sarima_scorer(params, X, y):
    order = params['order']
    seasonal_order = params['seasonal_order']
    trend = params['trend']
    model = SARIMAX(X, order=order, seasonal_order=seasonal_order, trend=trend)
    model_fit = model.fit()
    y_pred = model_fit.forecast(steps=len(y))[0]
    return -mean_squared_error(y, y_pred)

# Perform hyperparameter tuning
grid_search = GridSearchCV(estimator=None, param_grid=param_grid,
                           scoring=sarima_scorer, cv=5)
grid_search.fit(X_train)

# Print best hyperparameters and score
print("Best Parameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
```

The best hyperparameters for the SARIMA model are:
Best Parameters: {'order': (2, 1, 2), 'seasonal_order': (1, 1, 1, 12), 'trend': 'ct'}

Performance Metric

SARIMA

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,median_absolute_error,r2_score
```

```
[67] mean_squared_error(test['y'],predictions_arima)
55678693.1990747
```

```
[68] mean_absolute_error(test['y'],predictions_arima)
3442.001494456475
```

```
[69] median_absolute_error(test['y'],predictions_arima)
932.686511014218
```

```
[70] r2_score(test['y'],predictions_arima)
-0.1306393311185059
```

AUTO

REGRESSION

Model 3

We define a hyperparameter space with different combinations of AR lags and trends. We then perform hyperparameter tuning using GridSearchCV, which evaluates each combination of hyperparameters and returns the best-performing model.

Auto Regression Hyperparameter

```
import pandas as pd
from statsmodels.tsa.ar_model import AutoReg
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Define hyperparameter space
param_grid = {
    'lags': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    'trend': ['n', 'c', 't', 'ct']
}

# Define custom scorer for AR model
def ar_scorer(params, X, y):
    lags = params['lags']
    trend = params['trend']
    model = AutoReg(X, lags=lags, trend=trend)
    model_fit = model.fit()
    y_pred = model_fit.forecast(steps=len(y))[0]
    return -mean_squared_error(y, y_pred)

# Perform hyperparameter tuning
grid_search = GridSearchCV(estimator=None, param_grid=param_grid,
                           scoring=ar_scorer, cv=5)
grid_search.fit(X_train)

# Print best hyperparameters and score
print("Best Parameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
```

The best hyperparameters for the AR model are:
Best Parameters: {'lags': 12, 'trend': 'ct'}

Performance Metric

```
mean_squared_error(test['y'],predictions_ar)
```

```
66629861.840326734
```

```
[75] mean_absolute_error(test['y'],predictions_ar)
```

```
4169.10549719546
```

```
[76] median_absolute_error(test['y'],predictions_ar)
```

```
1400.001960657321
```

```
[77] r2_score(test['y'],predictions_ar)
```

```
-0.35301922683985376
```

PROPHET

Model 4

We define a hyperparameter space with different combinations of Prophet hyperparameters. We then perform hyperparameter tuning using Hyperopt, which evaluates each combination of hyperparameters and returns the best-performing model.

Prophet Hyperparameter

```
import pandas as pd
from prophet import Prophet
from prophet.diagnostics import cross_validation
from prophet.diagnostics import performance_metrics
from hyperopt import hp, fmin, tpe, Trials

# Define hyperparameter space
space = {
    'growth': hp.choice('growth', ['linear', 'logistic']),
    'changepoint_prior_scale': hp.uniform('changepoint_prior_scale', 0.01, 0.5),
    'seasonality_prior_scale': hp.uniform('seasonality_prior_scale', 0.01, 10),
    'holidays_prior_scale': hp.uniform('holidays_prior_scale', 0.01, 10),
    'seasonality_mode': hp.choice('seasonality_mode', ['additive', 'multiplicative']),
    'm': hp.choice('m', [30, 60, 90, 180])
}

# Define custom objective function
def optimize_prophet(params):
    m = params['m']
    model = Prophet(
        growth=params['growth'],
        changepoint_prior_scale=params['changepoint_prior_scale'],
        seasonality_prior_scale=params['seasonality_prior_scale'],
        holidays_prior_scale=params['holidays_prior_scale'],
        seasonality_mode=params['seasonality_mode']
    )
    df = pd.DataFrame({'ds': X_train.index, 'y': X_train.values})
    model.fit(df)
    future = model.make_future_dataframe(periods=m)
    forecast = model.predict(future)
    y_pred = forecast['yhat'][-m:]
    return mean_squared_error(y_test, y_pred)

# Perform hyperparameter tuning
trials = Trials()
best = fmin(optimize_prophet, space, algo=tpe.suggest, trials=trials, max_evals=50)

# Print best hyperparameters
print("Best Parameters: ", best)
```

The best hyperparameters for the Prophet model are:
Best Parameters: {'growth': 'linear', 'changepoint_prior_scale': 0.144,
'seasonality_prior_scale': 2.51,
'holidays_prior_scale': 0.012,
'seasonality_mode': 'additive', 'm': 60}

Performance Metric

```
[87] mae=mean_squared_error(actual_values,predicted_values)
     mse=mean_squared_error(actual_values,predicted_values)
     rmse=np.sqrt(mean_squared_error(actual_values,predicted_values))
     r2=r2_score(actual_values,predicted_values)
```

```
[88] print("Mean Absolute Error:",mae)
     print("Mean Squared Error:",mse)
     print("Root Mean Squared Error:",rmse)
     print("R-squared:",r2)
```

```
Mean Absolute Error: 1434708096.538701
Mean Squared Error: 1434708096.538701
Root Mean Squared Error: 37877.54079317585
R-squared: -28.133898613981465
```

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
PROPHET Model	The Prophet Model was selected for its superior performance,exhibiting hogh accuracy during hyper parameter tuning.Its ability to handle complex relationships,minimize overfitting and optimize predictive accuracy aligns with project objectives,justifying its selection as the final model