# **Machine Learning Project Documentation**

## **Model Refinement**

### **Overview**

The model refinement phase is a critical component of the machine learning pipeline, aimed at systematically improving model performance and robustness. In this project, the focus was on forecasting crude oil prices using a hybrid approach that combines ARIMA (AutoRegressive Integrated Moving Average) for trend modeling and LSTM (Long Short-Term Memory) neural networks for capturing complex residual patterns. The refinement process involved iterative experimentation, hyperparameter tuning, and feature engineering to address overfitting, enhance generalization, and ensure the model's applicability to real-world data.

### **Model Evaluation**

Initial model evaluation revealed that while the ARIMA model captured the main trend, residuals exhibited autocorrelation, indicating unmodeled structure. The LSTM, when trained on these residuals, initially showed signs of overfitting, with training loss decreasing rapidly but validation loss plateauing or increasing.

**Key Metrics and Visualizations:**

* **Root Mean Squared Error (RMSE)**: Measures the average magnitude of prediction errors.
* **Mean Absolute Percentage Error (MAPE)**: Normalized measure of prediction accuracy.
* **Loss Curves**: Plotted for training and validation to inspect overfitting and convergence.
* **Autocorrelation Plots**: Used to assess residual structure post-ARIMA.

These evaluations highlighted the need for regularization, improved feature engineering, and robust preprocessing.

### **Refinement Techniques**

**Regularization:**

* Increased dropout rates (up to 0.7) in LSTM layers.
* Applied L2 regularization (λ = 0.1) to LSTM weights.

**Model Simplification:**

* Reduced LSTM units and layers to decrease overfitting.

**Batch Normalization:**

* Introduced batch normalization layers to stabilize and accelerate training.

**Feature Engineering:**

* Capped outliers in 'Volume' using 5th and 95th percentiles (winsorization).
* Used ARIMA residuals and scaled 'Volume' as LSTM inputs.

**Robust NaN Handling:**

* Applied median imputation to missing values.

**Early Stopping:**

* Used with low patience to halt training when validation loss stagnated.

### **Hyperparameter Tuning**

**LSTM Hyperparameters:**

* Units: 8
* Dropout: 0.7
* L2 Regularization: 0.1
* Batch Size: 32
* Look-back Window: 90
* Early Stopping Patience: 2

**ARIMA Hyperparameters:**

* Selected using pmdarima.auto\_arima optimizing for AIC/BIC.

These changes led to a reduction in validation loss and enhanced training stability.

### **Cross-Validation**

To respect time-series structure:

* Used an 80/20 chronological train-test split.
* Considered walk-forward validation (not implemented due to constraints).

This ensured the model was evaluated on unseen future data.

### **Feature Selection**

Guided by domain knowledge and exploratory analysis:

* **Primary**: 'Close/Last' price
* **Secondary**: 'Volume' (after winsorization)

No automated feature selection was used due to the small, interpretable feature space.

### **Test Submission**

#### **Overview**

This phase ensured reliable model evaluation and deployment-readiness.

#### **Data Preparation for Testing**

* **Outlier Capping**: 5th/95th percentile applied to 'Volume'.
* **NaN Imputation**: Median filling.
* **Scaling**: Used training-fitted MinMaxScaler.
* **Sequence Creation**: Used consistent look-back windows.

#### **Model Application**

# Apply ARIMA to test set  
arima\_pred\_test = arima\_model.predict(start=split\_idx, end=len(df)-1)  
residuals\_test = test\_close - arima\_pred\_test  
volume\_test\_aligned = test\_volume  
  
test\_features = np.column\_stack([residuals\_test, volume\_test\_aligned])  
scaled\_test, \_, \_ = preprocess(train\_features, test\_features)  
X\_test, y\_test = create\_sequences(scaled\_test)  
lstm\_pred\_scaled = model.predict(X\_test)  
lstm\_pred = scaler.inverse\_transform(np.column\_stack([lstm\_pred\_scaled, np.zeros\_like(lstm\_pred\_scaled)]))[:,0]  
final\_pred = arima\_pred\_test[LOOK\_BACK:] + lstm\_pred

#### **Test Metrics**

Performance evaluated using:

* **RMSE**: Average prediction error magnitude.
* **MAPE**: Accuracy percentage.

rmse = np.sqrt(mean\_squared\_error(actual, final\_pred))  
mape = mean\_absolute\_percentage\_error(actual, final\_pred)

The test metrics aligned closely with validation, indicating good generalization.

### **Model Deployment**

* **LSTM**: Saved in HDF5 format (lstm\_model.h5)
* **ARIMA**: Saved as pickle (arima\_model.pkl)
* **Scaler**: Saved for consistent preprocessing

**Deployment:** FastAPI-based REST API deployed on Render.com or DigitalOcean.

* /train: Trains and saves both models
* /predict: Returns forecast from saved models

### **Code Implementation**

Modular codebase includes:

* **Data Loading/Preprocessing**: load\_data, preprocess
* **Sequence Creation**: create\_sequences
* **Model Training/Predicting**: train\_and\_predict, predict\_next
* Includes logging, exception handling, and clear inline comments.

### **Conclusion**

Refinement significantly improved model generalization by addressing overfitting and enhancing feature engineering. The test submission confirmed the model’s robustness on unseen data, and the deployment strategy supports reproducible, real-time forecasts. Challenges like outliers, missing data, and hybrid alignment were systematically handled, resulting in a production-ready model.

### **References**

* FastAPI: <https://fastapi.tiangolo.com/>
* Keras: <https://keras.io/>
* pmdarima: <https://alkaline-ml.com/pmdarima/>
* scikit-learn: <https://scikit-learn.org/>
* Deployment Guides: Render.com, DigitalOcean
* Project Codebase and Documentation