Model Optimization and Tuning Phase Template

Date	10 JULY 2024
Team ID	SWTID1720111029
Project Title	Unveiling Climate Change Dynamics through Earth Surface Temperature Analysis
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining neural network models for 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
Model 1 **RNN**	Hyperparam1: optimizer Description: Optimizer for the model training. Hyperparam2: dropout_rate Description: Dropout rate applied after each RNN layer. Hyperparam3: units Description: Number of units/neurons in each SimpleRNN layer.

	`` def create_rnn_model(optimizer='adam', dropout_rate=0.2, units=100): model = Sequential() model.add(Input(shape=(X_train_scaled.shape[1], X_train_scaled.shape[2]))) model.add(SimpleRNN(units, activation='relu', return_sequences=True)) model.add(Dropout(dropout_rate)) model.add(SimpleRNN(units, activation='relu')) model.add(Dropout(dropout_rate)) model.add(Dense(1)) model.compile(optimizer=optimizer, loss='mse') return model model = KerasRegressor(build_fn=create_rnn_model, verbose=0) param_grid = { 'optimizer': ['adam', 'rmsprop'], 'dropout_rate': [0.1, 0.2, 0.3], 'units': [50, 100, 150] } random_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=10, cv=3, verbose=2, random_state=42) random_search.fit(X_train_scaled, y_train_scaled) print("Best parameters found: ", random_search.best_params_) print("Best CV score: ", random_search.best_score_) best_model = random_search.best_estimator_ test_loss = best_model.score(X_test_scaled, y_test_scaled) print("Test loss: ", test_loss) ```
Model 2 **GRU**	Hyperparam1: optimizer Description: Optimizer for the model training. Hyperparam2: dropout_rate Description: Dropout rate applied after each GRU layer. Hyperparam3: units Description: Number of units/neurons in each GRU layer.

```python
# Define a function to create the GRU model
def create_gru_model(optimizer='adam', dropout_rate=0.2, units=100):
    model = Sequential()
    model.add(Input(shape=(X_train_scaled.shape[1], X_train_scaled.shape[2])))
    model.add(GRU(units, activation='relu', return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(GRU(units, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))
    model.compile(optimizer=optimizer, loss='mse')
    return model

model = KerasRegressor(build_fn=create_gru_model, verbose=0)

param_grid = {
    'optimizer': ['adam', 'rmsprop'],
    'dropout_rate': [0.1, 0.2, 0.3],
    'units': [50, 100, 150]
}

random_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
                                   n_iter=10, cv=3, verbose=2, random_state=42)

random_search.fit(X_train_scaled, y_train_scaled)

print("Best parameters found: ", random_search.best_params_)
print("Best CV score: ", random_search.best_score_)

best_model = random_search.best_estimator_
test_loss = best_model.score(X_test_scaled, y_test_scaled)
print("Test loss: ", test_loss)
```

Model 3 **LSTM**	units_1, units_2, units_3: Number of units in the LSTM layers. These parameters control the complexity and capacity of the LSTM layers. dropout_1, dropout_2, dropout_3: Dropout rate applied to the LSTM layers. Dropout is used for regularization, preventing overfitting by randomly setting a fraction of input units to zero at each update during training. dense_units: Number of units in the dense (fully connected) layer. This parameter determines the size of the hidden layer in the network after the LSTM layers.

dropout_dense: Dropout rate applied to the dense layer. Similar to LSTM dropout, this parameter helps prevent overfitting in the dense layer.

learning_rate: Learning rate for the Adam optimizer. This parameter controls the step size taken during optimization and affects the speed and stability of training.

```
class LSTMHyperModel(HyperModel):
    def build(self, hp):
        model = Sequential()
        model.add(Input(shape=(X_train_scaled.shape[1], 1)))
        model.add(Bidirectional(LSTM(units=hp.Int('units_1', min_value=32, max_value=256, step=32), return_sequences=True)))
        model.add(Dropout(rate=hp.Float('dropout_1', min_value=0.1, max_value=0.5, step=0.1)))
        model.add(Bidirectional(LSTM(units=hp.Int('units_2', min_value=32, max_value=256, step=32), return_sequences=True)))
        model.add(Dropout(rate=hp.Float('dropout_2', min_value=0.1, max_value=0.5, step=0.1)))
        model.add(Bidirectional(LSTM(units=hp.Int('units_3', min_value=32, max_value=256, step=32))))
        model.add(Dropout(rate=hp.Float('dropout_3', min_value=0.1, max_value=0.5, step=0.1)))
        model.add(Dense(units=hp.Int('dense_units', min_value=32, max_value=256, step=32), activation='relu'))
        model.add(Dropout(rate=hp.Float('dropout_dense', min_value=0.1, max_value=0.5, step=0.1)))
        model.add(Dense(1))
        model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', min_value=1e-4, max_value=1e-2, sampling='LOG')),
                      loss='mean_squared_error', metrics=['mae'])
        return model

from keras_tuner.tuners import RandomSearch

tuner = RandomSearch(
    LSTMHyperModel(),
    objective='val_loss',
    max_trials=10,
    executions_per_trial=2,
    directory='hyperparameter_tuning',
    project_name='lstm_temperature_prediction'
)

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

tuner.search(X_train_scaled, y_train_scaled, epochs=50, validation_data=(X_test_scaled, y_test_scaled), callbacks=[early_stopping], verbose=2)

best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

best_model = tuner.hypermodel.build(best_hps)

best_model.summary()
```

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Model 3	**Model 3**, the LSTM-based model with hyperparameter tuning, was chosen as the final optimized model for several reasons: • **Bidirectional LSTM Layers**: Model 3 utilizes bidirectional LSTM layers, which can capture both past and future context in sequences, potentially improving the model's ability to understand temporal dependencies in the temperature data. • **Dropout Regularization:** Dropout layers are incorporated after each LSTM layer, helping to prevent overfitting by randomly dropping neurons during training. • **Dense Layers**: It includes dense layers with ReLU activation, which can introduce non-linearity and potentially capture complex patterns in the data. • **Hyperparameter Tuning:** The use of keras_tuner allows for systematic exploration of hyperparameters like the number of units in LSTM layers, dropout rates, and learning rates of the optimizer. This tuning process helps find configurations that minimize validation loss, making the model more robust and less likely to overfit. • **Validation Strategy:** The model is tuned based on validation loss, ensuring that the selected hyperparameters generalize well to unseen data (test set). • **Early Stopping:** The EarlyStopping callback is used during hyperparameter tuning to prevent overfitting by stopping training when

	validation loss stops improving, ensuring the final model is not tuned excessively to the training data. Overall, Model 3 combines LSTM's ability to handle long-term dependencies with a systematic approach to hyperparameter tuning, making it a stronger candidate for optimized performance compared to Models 1 and 2, which lack the systematic hyperparameter optimization process.