# BiLSTM Model Documentation

## 1. Model Overview

The model is designed to predict **review ratings (1–5 stars)** from Amazon Fine Food Reviews text. It uses a **Bidirectional LSTM (BiLSTM)** architecture to capture the sequential and contextual information in the reviews.

**Why BiLSTM:**

- Reads text **forward and backward**, capturing context from both directions.
- Handles long and complex sentences better than traditional models (like TF-IDF + Logistic Regression).

---

## 2. Data Preprocessing

- **Text Cleaning:** Removed punctuation, special characters, and converted text to lowercase.
- **Tokenization:** Converted each review into a sequence of word indices.
- **Padding Sequences:** Ensured all sequences have the same length for input to the BiLSTM.
- **Derived Features:**
    - ReviewLength – number of words in each review
    - stopword_count – number of stopwords in each review

---

## 3. Model Architecture

1. **Embedding Layer** – Converts each word index into a dense vector of fixed size (captures semantic meaning).
2. **Bidirectional LSTM Layer** – Reads sequences in both forward and backward directions; includes dropout for regularization.
3. **Dense Layer(s)** – Fully connected layer(s) to output predicted rating.
4. **Output Layer** – Softmax activation for multi-class classification (ratings 1–5).

#### Hyperparameter Tuning

To optimize the BiLSTM model for the imbalanced Amazon Fine Food Reviews dataset, Keras Tuner was used with a Random Search strategy. The key goal was to maximize validation accuracy by exploring different combinations of important model hyperparameters.

**Hyperparameters Tuned:**

| Hyperparameter | Values Tested | Description |
| --- | --- | --- |
| Embedding Dimension | 50, 100, 200 | Size of the dense vector representing each word in the embedding layer |
| LSTM Units | 64, 128 | Number of hidden units in the BiLSTM layer |
| LSTM Dropout | 0.2 – 0.5 (step 0.1) | Dropout applied to the BiLSTM layer to prevent overfitting |
| Dense Units | 64, 128 | Number of neurons in the fully connected dense layer |
| Dense Dropout | 0.2 – 0.5 (step 0.1) | Dropout applied after the dense layer for regularization |

**Tuning Setup:**

- RandomSearch was chosen to explore a random subset of the hyperparameter space efficiently.
- Max Trials: 3 (for faster tuning; can be increased for more exhaustive search)
- Executions per Trial: 1
- Objective: Maximize validation accuracy on the validation set.

**Outcome / Notes:**

- The tuner automatically evaluated different combinations of embedding size, LSTM units, and dropout rates.
- The best hyperparameters were selected based on the highest validation accuracy, balancing performance and model complexity.
- This tuned BiLSTM model was then used for final training and evaluation on the imbalanced dataset.

---

## 4. Training & Evaluation

- **Dataset Split:** Training, validation, and test sets
- **Evaluation Metrics:** Accuracy, Precision, Recall, F1-score
- **Observations:**
    - Training and validation loss decreased steadily.
    - The model performs better than classical ML models due to context capture in BiLSTM.

---

## 5. Tokenization & Embedding Summary

- **Tokenization:** Converts text into sequences of integers (word indices).
- **Embedding Layer:** Maps each word index to a dense vector, allowing the model to learn **semantic relationships** between words.

---

## 6. Advantages

- Captures sequential context in reviews.
- Handles longer texts with complex patterns.
- More robust to imbalanced rating distribution than traditional ML models.

---

## 7. Final Notes

This BiLSTM model serves as a **production-ready text rating predictor** for Amazon Fine Food Reviews. It can be further optimized by tuning hyperparameters, adding more layers, or using pretrained embeddings (e.g., GloVe, Word2Vec).

## Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

import keras_tuner as kt

import tensorflow as tf


# 1. Load your dataset

# imbalanced_df has 'Review_text' and 'Rating'
```

```
X = imbalanced_df['Review_text'].astype(str).values
y = imbalanced_df['Rating'].values

# Encode labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)  # now labels: 0,1,2,3,4



# 2. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, stratify=y_encoded, random_state=42
)

# 3. Tokenization & Padding
max_words = 20000      # max words in vocabulary
max_len = 100          # max length of each review (reduces steps per epoch)

tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding='post',
truncating='post')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post',
truncating='post')

# 4. Build BiLSTM model function for hyperparameter tuning
def build_bilstm_model(hp):
```

```
    model = Sequential()
    model.add(Embedding(input_dim=max_words, output_dim=hp.Choice("embed_dim",
[50, 100, 200]), input_length=max_len))
    model.add(Bidirectional(LSTM(units=hp.Choice("lstm_units", [64, 128]),
dropout=hp.Float("dropout", 0.2, 0.5, step=0.1))))
    model.add(Dense(units=hp.Choice("dense_units", [64, 128]), activation='relu'))
    model.add(Dropout(hp.Float("dense_dropout", 0.2, 0.5, step=0.1)))
    model.add(Dense(5, activation='softmax'))  # 5 classes

    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

# 5. Hyperparameter Tuning
tuner = kt.RandomSearch(
    build_bilstm_model,
    objective='val_accuracy',
    max_trials=3,           # 3 trials for fast tuning
    executions_per_trial=1,
    directory='bilstm_tuning',
    project_name='imbalanced_reviews'
)

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

```
# 6. Run tuner
tuner.search(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=15,             # keep 15 epochs
    batch_size=128,        # increase batch size to reduce steps per epoch
    callbacks=[early_stop],
    verbose=1
)

# 7. Get best model
best_model = tuner.get_best_models(num_models=1)[0]

# Evaluate on test set
test_loss, test_acc = best_model.evaluate(X_test_pad, y_test, verbose=1)
print("Test Accuracy:", test_acc)
```