**Imports and Conf**

# === IMPORTS ===

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import warnings

from scipy import stats

from sklearn.metrics import roc\_curve

from sklearn.preprocessing import StandardScaler, RobustScaler

from sklearn.metrics import (

accuracy\_score, precision\_score, recall\_score, f1\_score,

roc\_auc\_score, confusion\_matrix, mean\_squared\_error, r2\_score

)

from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.optimizers import Adam

warnings.filterwarnings('ignore')

np.random.seed(42)

tf.random.set\_seed(42)

**Data Loading**

# === DATA LOADING FUNCTIONS ===

def load\_stock\_data(file\_path):

"""Load stock price data."""

print(f"Loading stock data from {file\_path}...")

try:

df = pd.read\_csv(file\_path)

df['date'] = pd.to\_datetime(df['Local Date'])

df = df.groupby('date').agg({

'Close': 'last', 'Open': 'first',

'High': 'max', 'Low': 'min', 'Volume': 'sum'

}).reset\_index()

df['Return'] = df['Close'].pct\_change().fillna(0)

df['Direction'] = (df['Return'] > 0).astype(int)

return df

except Exception as e:

print(f"Error loading stock data: {e}")

return pd.DataFrame()

def load\_sentiment\_data(file\_path):

"""Load sentiment data from FinBERT outputs."""

print(f"Loading sentiment data from {file\_path}...")

try:

df = pd.read\_csv(file\_path, encoding='utf-8')

if 'Publish date' in df.columns:

df['date'] = pd.to\_datetime(df['Publish date'])

else:

date\_col = [col for col in df.columns if 'date' in col.lower()][0]

df['date'] = pd.to\_datetime(df[date\_col])

sentiment\_col = [col for col in df.columns if 'sentiment' in col.lower()][0]

daily\_sentiment = df.groupby(df['date'].dt.date).agg({

sentiment\_col: ['mean', 'std', 'count']

}).reset\_index()

daily\_sentiment.columns = ['date', 'sentiment\_mean', 'sentiment\_std', 'sentiment\_count']

daily\_sentiment['date'] = pd.to\_datetime(daily\_sentiment['date'])

daily\_sentiment.fillna(0, inplace=True)

return daily\_sentiment

except Exception as e:

print(f"Error loading sentiment data: {e}")

return pd.DataFrame()

**Merge Stock and Sentiment Data**

def merge\_data(stock\_df, sentiment\_df):

"""Soft merge by nearest date within 1 day."""

if stock\_df.empty or sentiment\_df.empty:

print("Empty data, cannot merge.")

return pd.DataFrame()

stock\_df['date'] = pd.to\_datetime(stock\_df['date'])

sentiment\_df['date'] = pd.to\_datetime(sentiment\_df['date'])

stock\_df = stock\_df.sort\_values('date')

sentiment\_df = sentiment\_df.sort\_values('date')

merged = pd.merge\_asof(

stock\_df, sentiment\_df,

on='date', direction='backward',

tolerance=pd.Timedelta('1D')

)

merged.dropna(subset=['sentiment\_mean'], inplace=True)

print(f"Merged data: {merged.shape[0]} rows.")

return merged

**Feature Engineering**

# === FEATURE CREATION ===

def create\_features(df):

"""Create financial and sentiment features."""

if df.empty:

return df

df = df.copy()

df['price\_range'] = (df['High'] - df['Low']) / df['Open']

df['price\_change'] = (df['Close'] - df['Open']) / df['Open']

windows = [3, 5]

for w in windows:

df[f'ma\_{w}'] = df['Close'].rolling(w).mean()

df[f'volatility\_{w}'] = df['Return'].rolling(w).std()

df[f'volume\_change\_{w}'] = df['Volume'] / df['Volume'].rolling(w).mean()

df[f'sentiment\_ma\_{w}'] = df['sentiment\_mean'].rolling(w).mean()

df[f'sentiment\_volatility\_{w}'] = df['sentiment\_mean'].rolling(w).std()

df['sentiment\_momentum'] = df['sentiment\_mean'].diff()

df['sentiment\_price\_impact'] = df['sentiment\_mean'] \* df['price\_range']

df.fillna(0, inplace=True)

return df

**Prepare Data for Machine Learning and LSTM**

def prepare\_data(df, target\_col='Direction', test\_size=0.3, seq\_length=3):

"""Prepare ML and short-term LSTM data."""

if len(df) < 5:

print("Too few rows after feature creation, skipping.")

return {}, {}

features = [col for col in df.columns if col not in ['date', 'Close', 'Open', 'High', 'Low', 'Volume', 'Return', 'Direction']]

X = df[features].values

y = df[target\_col].values

split\_idx = int(len(X) \* (1 - test\_size))

X\_train, X\_test = X[:split\_idx], X[split\_idx:]

y\_train, y\_test = y[:split\_idx], y[split\_idx:]

scaler = RobustScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# ML data

ml\_data = {

'X\_train': X\_train\_scaled,

'X\_test': X\_test\_scaled,

'y\_train': y\_train,

'y\_test': y\_test

}

# LSTM short sequence data

if len(X\_train) >= seq\_length:

X\_train\_seq, y\_train\_seq = [], []

for i in range(len(X\_train\_scaled) - seq\_length):

X\_train\_seq.append(X\_train\_scaled[i:i+seq\_length])

y\_train\_seq.append(y\_train[i+seq\_length])

X\_test\_seq, y\_test\_seq = [], []

for i in range(len(X\_test\_scaled) - seq\_length):

X\_test\_seq.append(X\_test\_scaled[i:i+seq\_length])

y\_test\_seq.append(y\_test[i+seq\_length])

lstm\_data = {

'X\_train': np.array(X\_train\_seq),

'X\_test': np.array(X\_test\_seq),

'y\_train': np.array(y\_train\_seq),

'y\_test': np.array(y\_test\_seq),

'seq\_length': seq\_length,

'n\_features': X.shape[1]

}

else:

lstm\_data = {}

return ml\_data, lstm\_data

**Train Models**

# === MODEL TRAINING ===

def train\_xgboost(data):

model = xgb.XGBClassifier(

n\_estimators=50, max\_depth=3, learning\_rate=0.1,

objective='binary:logistic', random\_state=42

)

model.fit(data['X\_train'], data['y\_train'])

y\_pred = model.predict(data['X\_test'])

y\_proba = model.predict\_proba(data['X\_test'])[:, 1] # probability for class 1

return model, y\_pred, y\_proba

def train\_random\_forest(data):

model = RandomForestClassifier(

n\_estimators=50, max\_depth=3, class\_weight='balanced', random\_state=42

)

model.fit(data['X\_train'], data['y\_train'])

y\_pred = model.predict(data['X\_test'])

y\_proba = model.predict\_proba(data['X\_test'])[:, 1] # probability for class 1

return model, y\_pred, y\_proba

def train\_bi\_lstm(data, epochs=50):

if not data:

return None, []

model = Sequential()

model.add(Bidirectional(LSTM(8), input\_shape=(data['seq\_length'], data['n\_features'])))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=Adam(0.001), loss='binary\_crossentropy', metrics=['accuracy'])

model.fit(data['X\_train'], data['y\_train'], epochs=epochs, batch\_size=2, verbose=0)

y\_pred = model.predict(data['X\_test'])

y\_pred\_binary = (y\_pred > 0.5).astype(int).flatten()

return model, y\_pred\_binary

**Evaluation**

def evaluate(y\_true, y\_pred, y\_proba=None, model\_name="Model"):

print(f"\n{model\_name} Evaluation Metrics:")

acc = accuracy\_score(y\_true, y\_pred)

prec = precision\_score(y\_true, y\_pred, zero\_division=0)

rec = recall\_score(y\_true, y\_pred, zero\_division=0)

f1 = f1\_score(y\_true, y\_pred, zero\_division=0)

auc = roc\_auc\_score(y\_true, y\_proba) if y\_proba is not None else "N/A"

rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

r2 = r2\_score(y\_true, y\_pred)

correct\_preds = np.sum(y\_true == y\_pred)

n\_total = len(y\_true)

pval = stats.binomtest(correct\_preds, n\_total, p=0.5, alternative='greater').pvalue

# Format small p-values nicely

if pval < 1e-4:

pval\_str = f"{pval:.2e}"

else:

pval\_str = f"{pval:.4f}"

print(f"Accuracy: {acc:.4f}")

print(f"Precision: {prec:.4f}")

print(f"Recall: {rec:.4f}")

print(f"F1 Score: {f1:.4f}")

print(f"AUC: {auc if auc == 'N/A' else round(auc,4)}")

print(f"RMSE: {rmse:.4f}")

print(f"R²: {r2:.4f}")

print(f"p-value: {pval\_str}")

print("Confusion Matrix:")

print(confusion\_matrix(y\_true, y\_pred))

# === ROC Curve Plot ===

if y\_proba is not None:

fpr, tpr, thresholds = roc\_curve(y\_true, y\_proba)

plt.figure(figsize=(6,6))

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})')

plt.plot([0,1], [0,1], 'k--', label='Random Guess')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title(f'ROC Curve - {model\_name}')

plt.legend()

plt.grid(alpha=0.5)

plt.show()

**Main Function**

# === MAIN PROCESS ===

def process\_stock(stock\_file, sentiment\_file):

stock\_df = load\_stock\_data(stock\_file)

sentiment\_df = load\_sentiment\_data(sentiment\_file)

merged\_df = merge\_data(stock\_df, sentiment\_df)

featured\_df = create\_features(merged\_df)

ml\_data, lstm\_data = prepare\_data(featured\_df)

# Train XGBoost

xgb\_model, xgb\_pred, xgb\_proba = train\_xgboost(ml\_data)

evaluate(ml\_data['y\_test'], xgb\_pred, xgb\_proba, model\_name="XGBoost")

# Train Random Forest

rf\_model, rf\_pred, rf\_proba = train\_random\_forest(ml\_data)

evaluate(ml\_data['y\_test'], rf\_pred, rf\_proba, model\_name="Random Forest")

# Train LSTM (LSTM does not produce probability easily)

if lstm\_data:

lstm\_model, lstm\_pred = train\_bi\_lstm(lstm\_data)

evaluate(lstm\_data['y\_test'], lstm\_pred, model\_name="Bi-LSTM (no AUC)")

def main():

sentiment\_file = 'scraped\_articles.csv'

stock\_files = ['xli\_cleaned.csv', 'spy\_cleaned.csv', 'dia\_cleaned.csv']

for stock\_file in stock\_files:

print(f"\n{'='\*40}\nProcessing {stock\_file}...\n{'='\*40}")

process\_stock(stock\_file, sentiment\_file)

if \_*\_name*\_\_ == "\_*\_main*\_\_":

main()