CHAPTER 7

APPENDIX

```
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()  # Lowercase
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special chars
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])  # Lemmatize and remove stopwords
    return text

df['text'] = df['text'].apply(clean_text)  # Assuming 'text' column exists.
If not, replace with 'title' or relevant column
```

```
# Feature extraction
tfidf = TfidfVectorizer(max_features=5000)  # Limit to top 5000 features
X = tfidf.fit_transform(df['text'])
y = df['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Logistic Regression model
model = LogisticRegression(max_iter=1000)  # Increased max_iter for
convergence
model.fit(X_train, y_train)

# Predictions
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics for both training and test sets
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Predict on manual testing set (if available)
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        X_manual = tfidf.transform(test_df['cleaned_text'])
        manual_pred = model.predict(X_manual)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
```

```
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()  # Lowercase
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special chars
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])  # Lemmatize and remove stopwords
    return text

df['text'] = df['text'].apply(clean_text)  # Assuming 'text' column exists.
If not, replace with 'title' or relevant column

# Feature extraction
```

```
tfidf = TfidfVectorizer(max_features=5000)  # Limit to top 5000 features
X = tfidf.fit_transform(df['text'])
y = df['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Naive Bayes model (MultinomialNB)
model = MultinomialNB()
model.fit(X_train, y_train)

# Predictions
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics for both training and test sets
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Predict on manual testing set (if available)
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        X_manual = tfidf.transform(test_df['cleaned_text'])
        manual_pred = model.predict(X_manual)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
```

```
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC  # Changed to Support Vector Classifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()  # Lowercase
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special chars
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])  # Lemmatize and remove stopwords
    return text

df['text'] = df['text'].apply(clean_text)  # Assuming 'text' column exists.
If not, replace with 'title' or relevant column

# Feature extraction
tfidf = TfidfVectorizer(max_features=5000)  # Limit to top 5000 features
X = tfidf.fit_transform(df['text'])
```

```
y = df['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# SVM model (Support Vector Classifier)
model = SVC(kernel='linear', probability=True)  # Linear kernel works best
for text data
model.fit(X_train, y_train)

# Predictions
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics for both training and test sets
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Predict on manual testing set (if available)
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        X_manual = tfidf.transform(test_df['cleaned_text'])
        manual_pred = model.predict(X_manual)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
```

```
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
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, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()  # Ensure string and lowercase
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special chars
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])  # Lemmatize and remove stopwords
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
```

```
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Tokenization
max_words = 5000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train)

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

# Padding sequences
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# LSTM Model
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=128,
input_length=max_len))
model.add(LSTM(64, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(32))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

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

# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=3)

# Train model
history = model.fit(X_train_pad, y_train,
                    epochs=10,
                    batch_size=64,
                    validation_split=0.1,
                    callbacks=[early_stop])

# Predictions
train_pred = (model.predict(X_train_pad) > 0.5).astype("int32")
test_pred = (model.predict(X_test_pad) > 0.5).astype("int32")
```

```
# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Predict on manual testing set (if available)
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        test_pad = pad_sequences(test_seq, maxlen=max_len)
        manual_pred = (model.predict(test_pad) > 0.5).astype("int32")
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
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, LSTM, Dense, Dropout,
Bidirectional
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()  # Ensure string and lowercase
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special chars
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])  # Lemmatize and remove stopwords
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Tokenization
max_words = 10000  # Increased vocabulary size
max_len = 200      # Maximum sequence length

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)

# Padding sequences
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')

# LSTM Model Architecture
model = Sequential([
    Embedding(input_dim=max_words, output_dim=256, input_length=max_len),
    Bidirectional(LSTM(128, return_sequences=True)),
    Dropout(0.3),
    Bidirectional(LSTM(64)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

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

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

# Train model
history = model.fit(
    X_train_pad, y_train,
    epochs=15,
    batch_size=64,
    validation_split=0.1,
    callbacks=[early_stop],
    verbose=1
)

# Plot training history
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()

# Predictions
train_pred = (model.predict(X_train_pad) > 0.5).astype("int32")
test_pred = (model.predict(X_test_pad) > 0.5).astype("int32")

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Predict on manual testing set (if available)
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        test_pad = pad_sequences(test_seq, maxlen=max_len, padding='post',
truncating='post')
        manual_pred = (model.predict(test_pad) > 0.5).astype("int32")
        test_df['predicted_label'] = manual_pred
```

```
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import matplotlib.pyplot as plt
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text
```

```
df['cleaned_text'] = df['text'].apply(clean_text)

# Feature extraction
tfidf = TfidfVectorizer(max_features=10000)  # Using 10,000 features
X = tfidf.fit_transform(df['cleaned_text'])
y = df['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Hyperparameter tuning setup
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize Random Forest with basic parameters
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Randomized Search CV
rf_random = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=20,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# Fit the random search model
rf_random.fit(X_train, y_train)

# Best parameters
print(f"\nBest Parameters: {rf_random.best_params_}")

# Train final model with best parameters
best_rf = rf_random.best_estimator_

# Predictions
```

```
train_pred = best_rf.predict(X_train)
test_pred = best_rf.predict(X_test)

# Feature Importance
feature_importances = best_rf.feature_importances_
important_features = sorted(zip(tfidf.get_feature_names_out(),
feature_importances),
                            key=lambda x: x[1], reverse=True)[:20]

print("\nTop 20 Important Features:")
for feat, importance in important_features:
    print(f"{feat}: {importance:.4f}")

# Plot Feature Importance
plt.figure(figsize=(10, 6))
plt.barh([x[0] for x in important_features[::-1]], [x[1] for x in
important_features[::-1]])
plt.xlabel('Importance Score')
plt.title('Top 20 Important Features')
plt.tight_layout()
plt.show()

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        X_manual = tfidf.transform(test_df['cleaned_text'])
```

```
        manual_pred = best_rf.predict(X_manual)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()  # Ensure string and lowercase
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special chars
```

```
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])  # Lemmatize and remove stopwords
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Feature extraction
tfidf = TfidfVectorizer(max_features=10000)  # Increased to 10000 features
X = tfidf.fit_transform(df['cleaned_text'])
y = df['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize individual classifiers
log_clf = LogisticRegression(max_iter=1000, C=1.0)
nb_clf = MultinomialNB()
svm_clf = SVC(kernel='linear', probability=True, C=0.5)

# Create Voting Classifier (soft voting)
voting_clf = VotingClassifier(
    estimators=[
        ('lr', log_clf),
        ('nb', nb_clf),
        ('svm', svm_clf)
    ],
    voting='soft'  # Uses predicted probabilities
)

# Train voting classifier
voting_clf.fit(X_train, y_train)

# Predictions
train_pred = voting_clf.predict(X_train)
test_pred = voting_clf.predict(X_test)

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
```

```
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics for both training and test sets
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Individual classifier evaluation
print("\nIndividual Classifier Performance:")
for clf in (log_clf, nb_clf, svm_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"\n{clf.__class__.__name__} Accuracy: {accuracy_score(y_test,
y_pred):.4f}")

# Predict on manual testing set (if available)
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        X_manual = tfidf.transform(test_df['cleaned_text'])
        manual_pred = voting_clf.predict(X_manual)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

```
# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()  # Ensure string and lowercase
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special chars
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])  # Lemmatize and remove stopwords
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Feature extraction
tfidf = TfidfVectorizer(max_features=10000)  # Increased to 10000 features
X = tfidf.fit_transform(df['cleaned_text'])
y = df['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Define base estimators
base_estimators = [
    ('lr', LogisticRegression(max_iter=1000, C=1.0)),
    ('nb', MultinomialNB()),
    ('svm', SVC(kernel='linear', probability=True, C=0.5))
]

# Define meta-classifier (final estimator)
```

```
meta_classifier = LogisticRegression(max_iter=1000, C=0.8)

# Create Stacking Classifier
stacking_clf = StackingClassifier(
    estimators=base_estimators,
    final_estimator=meta_classifier,
    stack_method='auto',  # Uses predict_proba if available
    passthrough=False,    # Doesn't include original features
    cv=5                  # 5-fold cross-validation for stacking
)

# Train stacking classifier
stacking_clf.fit(X_train, y_train)

# Predictions
train_pred = stacking_clf.predict(X_train)
test_pred = stacking_clf.predict(X_test)

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics for both training and test sets
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Individual classifier evaluation
print("\nIndividual Classifier Performance:")
for name, clf in base_estimators:
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"\n{name.upper()} Accuracy: {accuracy_score(y_test, y_pred):.4f}")

# Meta-classifier performance (on stacked features)
print(f"\nMeta-classifier (LogisticRegression) performance on stacked
features")
```

```
# Predict on manual testing set (if available)
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        X_manual = tfidf.transform(test_df['cleaned_text'])
        manual_pred = stacking_clf.predict(X_manual)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
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, Conv1D, MaxPooling1D, LSTM,
Dense, Dropout, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.regularizers import l2

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news
```

```
# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Tokenization
max_words = 20000
max_len = 300

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)

# Padding sequences
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')

# Hybrid CNN-LSTM Model Architecture
model = Sequential([
    Embedding(input_dim=max_words, output_dim=256, input_length=max_len),

    # CNN Layers
    Conv1D(128, 5, activation='relu', kernel_regularizer=l2(0.01)),
    MaxPooling1D(2),
```

```
    Dropout(0.3),

    Conv1D(64, 3, activation='relu', kernel_regularizer=l2(0.01)),
    MaxPooling1D(2),
    Dropout(0.3),

    # LSTM Layers
    Bidirectional(LSTM(64, return_sequences=True,
kernel_regularizer=l2(0.01))),
    Dropout(0.3),

    Bidirectional(LSTM(32, kernel_regularizer=l2(0.01))),
    Dropout(0.3),

    # Dense Layers
    Dense(64, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.2),

    Dense(1, activation='sigmoid')
])

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

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy',
save_best_only=True, mode='max')

# Train model
history = model.fit(
    X_train_pad, y_train,
    epochs=15,
    batch_size=64,
    validation_split=0.1,
    callbacks=[early_stop, checkpoint],
    verbose=1
)

# Plot training history
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()

# Load best model
model.load_weights('best_model.h5')

# Predictions
train_pred = (model.predict(X_train_pad) > 0.5).astype("int32")
test_pred = (model.predict(X_test_pad) > 0.5).astype("int32")

# Evaluation metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
```

```
        test_pad = pad_sequences(test_seq, maxlen=max_len, padding='post',
truncating='post')
        manual_pred = (model.predict(test_pad) > 0.5).astype("int32")
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
```

```
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Custom transformer for hybrid features
class HybridFeatureExtractor(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.tfidf = TfidfVectorizer(max_features=10000)
        self.nb = MultinomialNB()
        self.lr = LogisticRegression(max_iter=1000)

    def fit(self, X, y):
        X_tfidf = self.tfidf.fit_transform(X)
        self.nb.fit(X_tfidf, y)
        self.lr.fit(X_tfidf, y)
        return self

    def transform(self, X):
        X_tfidf = self.tfidf.transform(X)
        nb_probs = self.nb.predict_proba(X_tfidf)
        lr_probs = self.lr.predict_proba(X_tfidf)
        return np.hstack([X_tfidf.toarray(), nb_probs, lr_probs])

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create hybrid model pipeline
hybrid_model = Pipeline([
    ('features', HybridFeatureExtractor()),
    ('classifier', LogisticRegression(max_iter=1000))  # Meta-classifier
])

# Train model
hybrid_model.fit(X_train, y_train)

# Predictions
train_pred = hybrid_model.predict(X_train)
test_pred = hybrid_model.predict(X_test)
```

```
# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")

# Naive Bayes alone
nb = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=10000)),
    ('classifier', MultinomialNB())
])
nb.fit(X_train, y_train)
nb_pred = nb.predict(X_test)
print(f"\nNaive Bayes Accuracy: {accuracy_score(y_test, nb_pred):.4f}")

# Logistic Regression alone
lr = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=10000)),
    ('classifier', LogisticRegression(max_iter=1000))
])
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, lr_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        manual_pred = hybrid_model.predict(test_df['cleaned_text'])
        test_df['predicted_label'] = manual_pred
```

```
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
```

```
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Custom transformer for hybrid features
class HybridFeatureExtractor(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.tfidf = TfidfVectorizer(max_features=10000)
        self.svm = SVC(kernel='linear', probability=True)
        self.lr = LogisticRegression(max_iter=1000)

    def fit(self, X, y):
        X_tfidf = self.tfidf.fit_transform(X)
        self.svm.fit(X_tfidf, y)
        self.lr.fit(X_tfidf, y)
        return self

    def transform(self, X):
        X_tfidf = self.tfidf.transform(X)
        svm_probs = self.svm.predict_proba(X_tfidf)
        lr_probs = self.lr.predict_proba(X_tfidf)
        return np.hstack([X_tfidf.toarray(), svm_probs, lr_probs])

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create hybrid model pipeline
hybrid_model = Pipeline([
    ('features', HybridFeatureExtractor()),
    ('classifier', LogisticRegression(max_iter=1000))  # Meta-classifier
])

# Train model
hybrid_model.fit(X_train, y_train)

# Predictions
train_pred = hybrid_model.predict(X_train)
test_pred = hybrid_model.predict(X_test)

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
```

```
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")

# SVM alone
svm = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=10000)),
    ('classifier', SVC(kernel='linear', probability=True))
])
svm.fit(X_train, y_train)
svm_pred = svm.predict(X_test)
print(f"\nSVM Accuracy: {accuracy_score(y_test, svm_pred):.4f}")

# Logistic Regression alone
lr = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=10000)),
    ('classifier', LogisticRegression(max_iter=1000))
])
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, lr_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        manual_pred = hybrid_model.predict(test_df['cleaned_text'])
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
```

```
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
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, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text
```

```
df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature Extraction - Two Pathways
# 1. TF-IDF for Logistic Regression
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# 2. Tokenization for LSTM
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Train Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_tfidf, y_train)

# Train LSTM
lstm_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    LSTM(64, return_sequences=True),
    Dropout(0.3),
    LSTM(32),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

lstm_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

```
lstm_model.fit(X_train_pad, y_train, epochs=10, batch_size=64,
validation_split=0.1,
              callbacks=[EarlyStopping(monitor='val_loss', patience=2)])

# Get predictions from both models
lr_train_pred = lr.predict_proba(X_train_tfidf)
lr_test_pred = lr.predict_proba(X_test_tfidf)

lstm_train_pred = lstm_model.predict(X_train_pad)
lstm_test_pred = lstm_model.predict(X_test_pad)

# Combine predictions (stack features)
X_train_combined = np.hstack([lr_train_pred, lstm_train_pred])
X_test_combined = np.hstack([lr_test_pred, lstm_test_pred])

# Meta-classifier (Logistic Regression)
meta_classifier = LogisticRegression(max_iter=1000)
meta_classifier.fit(X_train_combined, y_train)

# Final predictions
train_pred = meta_classifier.predict(X_train_combined)
test_pred = meta_classifier.predict(X_test_combined)

# Evaluation
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")
lr_pred = lr.predict(X_test_tfidf)
print(f"\nLogistic Regression Accuracy: {accuracy_score(y_test,
lr_pred):.4f}")
```

```
lstm_pred = (lstm_model.predict(X_test_pad) > 0.5).astype("int32")
print(f"LSTM Accuracy: {accuracy_score(y_test, lstm_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # TF-IDF features
        manual_tfidf = tfidf.transform(test_df['cleaned_text'])

        # LSTM features
        manual_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        manual_pad = pad_sequences(manual_seq, maxlen=max_len,
padding='post')

        # Get predictions
        lr_manual_pred = lr.predict_proba(manual_tfidf)
        lstm_manual_pred = lstm_model.predict(manual_pad)

        # Combine
        manual_combined = np.hstack([lr_manual_pred, lstm_manual_pred])

        # Final prediction
        manual_pred = meta_classifier.predict(manual_combined)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
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, Conv1D, GlobalMaxPooling1D,
Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature Extraction - Two Pathways
# 1. TF-IDF for Logistic Regression
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

```
# 2. Tokenization for CNN
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Train Logistic Regression
lr = LogisticRegression(max_iter=1000, C=0.8)
lr.fit(X_train_tfidf, y_train)

# Train CNN
cnn_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    Conv1D(128, 5, activation='relu'),
    GlobalMaxPooling1D(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

cnn_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
cnn_model.fit(X_train_pad, y_train, epochs=10, batch_size=64,
validation_split=0.1,
             callbacks=[EarlyStopping(monitor='val_loss', patience=2)])

# Get predictions from both models
lr_train_pred = lr.predict_proba(X_train_tfidf)
lr_test_pred = lr.predict_proba(X_test_tfidf)

cnn_train_pred = cnn_model.predict(X_train_pad)
cnn_test_pred = cnn_model.predict(X_test_pad)

# Combine predictions (stack features)
X_train_combined = np.hstack([lr_train_pred, cnn_train_pred])
X_test_combined = np.hstack([lr_test_pred, cnn_test_pred])
```

```
# Meta-classifier (Logistic Regression)
meta_classifier = LogisticRegression(max_iter=1000)
meta_classifier.fit(X_train_combined, y_train)

# Final predictions
train_pred = meta_classifier.predict(X_train_combined)
test_pred = meta_classifier.predict(X_test_combined)

# Evaluation
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")
lr_pred = lr.predict(X_test_tfidf)
print(f"\nLogistic Regression Accuracy: {accuracy_score(y_test,
lr_pred):.4f}")

cnn_pred = (cnn_model.predict(X_test_pad) > 0.5).astype("int32")
print(f"CNN Accuracy: {accuracy_score(y_test, cnn_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # TF-IDF features
        manual_tfidf = tfidf.transform(test_df['cleaned_text'])

        # CNN features
        manual_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
```

```
        manual_pad = pad_sequences(manual_seq, maxlen=max_len,
padding='post')

        # Get predictions
        lr_manual_pred = lr.predict_proba(manual_tfidf)
        cnn_manual_pred = cnn_model.predict(manual_pad)

        # Combine
        manual_combined = np.hstack([lr_manual_pred, cnn_manual_pred])

        # Final prediction
        manual_pred = meta_classifier.predict(manual_combined)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news
```

```
# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Custom transformer for hybrid features
class HybridFeatureExtractor(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.tfidf = TfidfVectorizer(max_features=10000)
        self.nb = MultinomialNB()
        self.svm = SVC(kernel='linear', probability=True)

    def fit(self, X, y):
        X_tfidf = self.tfidf.fit_transform(X)
        self.nb.fit(X_tfidf, y)
        self.svm.fit(X_tfidf, y)
        return self

    def transform(self, X):
        X_tfidf = self.tfidf.transform(X)
        nb_probs = self.nb.predict_proba(X_tfidf)
        svm_probs = self.svm.predict_proba(X_tfidf)
        return np.hstack([X_tfidf.toarray(), nb_probs, svm_probs])

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create hybrid model pipeline
hybrid_model = Pipeline([
    ('features', HybridFeatureExtractor()),
```

```
    ('classifier', SVC(kernel='linear', probability=True))  # Meta-classifier
])

# Train model
hybrid_model.fit(X_train, y_train)

# Predictions
train_pred = hybrid_model.predict(X_train)
test_pred = hybrid_model.predict(X_test)

# Calculate all metrics
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Print metrics
print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")

# Naive Bayes alone
nb = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=10000)),
    ('classifier', MultinomialNB())
])
nb.fit(X_train, y_train)
nb_pred = nb.predict(X_test)
print(f"\nNaive Bayes Accuracy: {accuracy_score(y_test, nb_pred):.4f}")

# SVM alone
svm = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=10000)),
    ('classifier', SVC(kernel='linear'))
])
svm.fit(X_train, y_train)
```

```
svm_pred = svm.predict(X_test)
print(f"SVM Accuracy: {accuracy_score(y_test, svm_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)
        manual_pred = hybrid_model.predict(test_df['cleaned_text'])
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
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, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news
```

```
# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature Extraction - Two Pathways
# 1. TF-IDF for Naive Bayes
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# 2. Tokenization for LSTM
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Train Naive Bayes
nb = MultinomialNB()
nb.fit(X_train_tfidf, y_train)
```

```
# Train LSTM
lstm_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    LSTM(64, return_sequences=True),
    Dropout(0.3),
    LSTM(32),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

lstm_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
lstm_model.fit(X_train_pad, y_train, epochs=10, batch_size=64,
validation_split=0.1,
              callbacks=[EarlyStopping(monitor='val_loss', patience=2)])

# Get predictions from both models
nb_train_pred = nb.predict_proba(X_train_tfidf)
nb_test_pred = nb.predict_proba(X_test_tfidf)

lstm_train_pred = lstm_model.predict(X_train_pad)
lstm_test_pred = lstm_model.predict(X_test_pad)

# Combine predictions (stack features)
X_train_combined = np.hstack([nb_train_pred, lstm_train_pred])
X_test_combined = np.hstack([nb_test_pred, lstm_test_pred])

# Meta-classifier (Logistic Regression)
from sklearn.linear_model import LogisticRegression
meta_classifier = LogisticRegression(max_iter=1000)
meta_classifier.fit(X_train_combined, y_train)

# Final predictions
train_pred = meta_classifier.predict(X_train_combined)
test_pred = meta_classifier.predict(X_test_combined)

# Evaluation
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
```

```
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")
nb_pred = nb.predict(X_test_tfidf)
print(f"\nNaive Bayes Accuracy: {accuracy_score(y_test, nb_pred):.4f}")

lstm_pred = (lstm_model.predict(X_test_pad) > 0.5).astype("int32")
print(f"LSTM Accuracy: {accuracy_score(y_test, lstm_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # TF-IDF features
        manual_tfidf = tfidf.transform(test_df['cleaned_text'])

        # LSTM features
        manual_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        manual_pad = pad_sequences(manual_seq, maxlen=max_len,
padding='post')

        # Get predictions
        nb_manual_pred = nb.predict_proba(manual_tfidf)
        lstm_manual_pred = lstm_model.predict(manual_pad)

        # Combine
        manual_combined = np.hstack([nb_manual_pred, lstm_manual_pred])

        # Final prediction
        manual_pred = meta_classifier.predict(manual_combined)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
```

```
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
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, Conv1D, GlobalMaxPooling1D,
Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text
```

```
df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature Extraction - Two Pathways
# 1. TF-IDF for Naive Bayes
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# 2. Tokenization for CNN
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Train Naive Bayes
nb = MultinomialNB()
nb.fit(X_train_tfidf, y_train)

# Train CNN
cnn_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    Conv1D(128, 5, activation='relu'),
    GlobalMaxPooling1D(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

cnn_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

```
cnn_model.fit(X_train_pad, y_train, epochs=10, batch_size=64,
validation_split=0.1,
             callbacks=[EarlyStopping(monitor='val_loss', patience=2)])

# Get predictions from both models
nb_train_pred = nb.predict_proba(X_train_tfidf)
nb_test_pred = nb.predict_proba(X_test_tfidf)

cnn_train_pred = cnn_model.predict(X_train_pad)
cnn_test_pred = cnn_model.predict(X_test_pad)

# Combine predictions (stack features)
X_train_combined = np.hstack([nb_train_pred, cnn_train_pred])
X_test_combined = np.hstack([nb_test_pred, cnn_test_pred])

# Meta-classifier (Logistic Regression)
from sklearn.linear_model import LogisticRegression
meta_classifier = LogisticRegression(max_iter=1000)
meta_classifier.fit(X_train_combined, y_train)

# Final predictions
train_pred = meta_classifier.predict(X_train_combined)
test_pred = meta_classifier.predict(X_test_combined)

# Evaluation
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")
nb_pred = nb.predict(X_test_tfidf)
print(f"\nNaive Bayes Accuracy: {accuracy_score(y_test, nb_pred):.4f}")
```

```
cnn_pred = (cnn_model.predict(X_test_pad) > 0.5).astype("int32")
print(f"CNN Accuracy: {accuracy_score(y_test, cnn_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # TF-IDF features
        manual_tfidf = tfidf.transform(test_df['cleaned_text'])

        # CNN features
        manual_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        manual_pad = pad_sequences(manual_seq, maxlen=max_len,
padding='post')

        # Get predictions
        nb_manual_pred = nb.predict_proba(manual_tfidf)
        cnn_manual_pred = cnn_model.predict(manual_pad)

        # Combine
        manual_combined = np.hstack([nb_manual_pred, cnn_manual_pred])

        # Final prediction
        manual_pred = meta_classifier.predict(manual_combined)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
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, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature Extraction - Two Pathways
# 1. TF-IDF for SVM
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

```
# 2. Tokenization for LSTM
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Train SVM
svm = SVC(kernel='linear', probability=True, C=0.5)
svm.fit(X_train_tfidf, y_train)

# Train LSTM
lstm_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    LSTM(64, return_sequences=True),
    Dropout(0.3),
    LSTM(32),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

lstm_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
lstm_model.fit(X_train_pad, y_train, epochs=10, batch_size=64,
validation_split=0.1,
              callbacks=[EarlyStopping(monitor='val_loss', patience=2)])

# Get predictions from both models
svm_train_pred = svm.predict_proba(X_train_tfidf)
svm_test_pred = svm.predict_proba(X_test_tfidf)

lstm_train_pred = lstm_model.predict(X_train_pad)
lstm_test_pred = lstm_model.predict(X_test_pad)

# Combine predictions (stack features)
X_train_combined = np.hstack([svm_train_pred, lstm_train_pred])
X_test_combined = np.hstack([svm_test_pred, lstm_test_pred])

# Meta-classifier (Logistic Regression)
```

```
from sklearn.linear_model import LogisticRegression
meta_classifier = LogisticRegression(max_iter=1000)
meta_classifier.fit(X_train_combined, y_train)

# Final predictions
train_pred = meta_classifier.predict(X_train_combined)
test_pred = meta_classifier.predict(X_test_combined)

# Evaluation
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")
svm_pred = svm.predict(X_test_tfidf)
print(f"\nSVM Accuracy: {accuracy_score(y_test, svm_pred):.4f}")

lstm_pred = (lstm_model.predict(X_test_pad) > 0.5).astype("int32")
print(f"LSTM Accuracy: {accuracy_score(y_test, lstm_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # TF-IDF features
        manual_tfidf = tfidf.transform(test_df['cleaned_text'])

        # LSTM features
        manual_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        manual_pad = pad_sequences(manual_seq, maxlen=max_len,
padding='post')
```

```
        # Get predictions
        svm_manual_pred = svm.predict_proba(manual_tfidf)
        lstm_manual_pred = lstm_model.predict(manual_pad)

        # Combine
        manual_combined = np.hstack([svm_manual_pred, lstm_manual_pred])

        # Final prediction
        manual_pred = meta_classifier.predict(manual_combined)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
import pandas as pd
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
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, Conv1D, GlobalMaxPooling1D,
Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Load and prepare data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
```

```
true_df['label'] = 1   # 1 for true news

# Combine datasets and shuffle
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X = df['cleaned_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature Extraction - Two Pathways
# 1. TF-IDF for SVM
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# 2. Tokenization for CNN
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Train SVM
svm = SVC(kernel='linear', probability=True, C=0.5)
```

```
svm.fit(X_train_tfidf, y_train)

# Train CNN
cnn_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    Conv1D(128, 5, activation='relu'),
    GlobalMaxPooling1D(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

cnn_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
cnn_model.fit(X_train_pad, y_train, epochs=10, batch_size=64,
validation_split=0.1,
             callbacks=[EarlyStopping(monitor='val_loss', patience=2)])

# Get predictions from both models
svm_train_pred = svm.predict_proba(X_train_tfidf)
svm_test_pred = svm.predict_proba(X_test_tfidf)

cnn_train_pred = cnn_model.predict(X_train_pad)
cnn_test_pred = cnn_model.predict(X_test_pad)

# Combine predictions (stack features)
X_train_combined = np.hstack([svm_train_pred, cnn_train_pred])
X_test_combined = np.hstack([svm_test_pred, cnn_test_pred])

# Meta-classifier (Logistic Regression)
from sklearn.linear_model import LogisticRegression
meta_classifier = LogisticRegression(max_iter=1000)
meta_classifier.fit(X_train_combined, y_train)

# Final predictions
train_pred = meta_classifier.predict(X_train_combined)
test_pred = meta_classifier.predict(X_test_combined)

# Evaluation
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
```

```
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

print_metrics(y_train, train_pred, "Training")
print_metrics(y_test, test_pred, "Test")

# Compare individual models
print("\nIndividual Model Performance:")
svm_pred = svm.predict(X_test_tfidf)
print(f"\nSVM Accuracy: {accuracy_score(y_test, svm_pred):.4f}")

cnn_pred = (cnn_model.predict(X_test_pad) > 0.5).astype("int32")
print(f"CNN Accuracy: {accuracy_score(y_test, cnn_pred):.4f}")

# Predict on manual testing set
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # TF-IDF features
        manual_tfidf = tfidf.transform(test_df['cleaned_text'])

        # CNN features
        manual_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        manual_pad = pad_sequences(manual_seq, maxlen=max_len,
padding='post')

        # Get predictions
        svm_manual_pred = svm.predict_proba(manual_tfidf)
        cnn_manual_pred = cnn_model.predict(manual_pad)

        # Combine
        manual_combined = np.hstack([svm_manual_pred, cnn_manual_pred])

        # Final prediction
        manual_pred = meta_classifier.predict(manual_combined)
        test_df['predicted_label'] = manual_pred
        print("\nPredictions on manual_testing.csv:")
        print(test_df[['text', 'predicted_label']].head())
    else:
```

```
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
# Installation
!pip install tensorflow scikit-learn numpy pandas nltk
nltk.download('stopwords')
nltk.download('wordnet')

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Conv1D,
GlobalMaxPooling1D, Dense, Dropout, Concatenate
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from sklearn.feature_extraction.text import TfidfVectorizer

# Load and preprocess data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1  # 1 for true news
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text cleaning
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
```

```
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df['cleaned_text'], df['label'], test_size=0.2, random_state=42
)

# Two feature extraction methods:
# 1. For CNN (sequence data)
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# 2. For traditional models (TF-IDF)
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# CNN Model
def build_cnn_model():
    input_layer = Input(shape=(max_len,))
    embedding = Embedding(max_words, 128)(input_layer)

    conv1 = Conv1D(128, 3, activation='relu')(embedding)
    pool1 = GlobalMaxPooling1D()(conv1)

    conv2 = Conv1D(128, 4, activation='relu')(embedding)
    pool2 = GlobalMaxPooling1D()(conv2)

    conv3 = Conv1D(128, 5, activation='relu')(embedding)
    pool3 = GlobalMaxPooling1D()(conv3)

    merged = Concatenate()([pool1, pool2, pool3])
```

```
    dense = Dense(64, activation='relu')(merged)
    dropout = Dropout(0.5)(dense)
    output = Dense(1, activation='sigmoid')(dropout)

    model = Model(inputs=input_layer, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# Train CNN separately
print("Training CNN model...")
cnn_model = build_cnn_model()
cnn_model.fit(X_train_pad, y_train, epochs=3, batch_size=32,
validation_split=0.1)

# Get CNN predictions
X_train_cnn_pred = cnn_model.predict(X_train_pad)
X_test_cnn_pred = cnn_model.predict(X_test_pad)

# Initialize traditional models
log_clf = LogisticRegression(max_iter=1000, C=1.0)
nb_clf = MultinomialNB()
svm_clf = SVC(kernel='linear', probability=True, C=0.5)

# Train traditional models on TF-IDF features
print("\nTraining traditional models...")
log_clf.fit(X_train_tfidf, y_train)
nb_clf.fit(X_train_tfidf, y_train)
svm_clf.fit(X_train_tfidf, y_train)

# Create ensemble predictions
def ensemble_predict(X_tfidf, X_pad):
    # Get predictions from all models
    cnn_pred = cnn_model.predict(X_pad).flatten()
    lr_pred = log_clf.predict_proba(X_tfidf)[:, 1]
    nb_pred = nb_clf.predict_proba(X_tfidf)[:, 1]
    svm_pred = svm_clf.predict_proba(X_tfidf)[:, 1]

    # Average probabilities (soft voting)
    avg_proba = (cnn_pred + lr_pred + nb_pred + svm_pred) / 4
    return (avg_proba > 0.5).astype(int)

# Evaluate ensemble
test_pred = ensemble_predict(X_test_tfidf, X_test_pad)
```

```
print("\nEnsemble Test Metrics:")
print(f"Accuracy: {accuracy_score(y_test, test_pred):.4f}")
print(f"Precision: {precision_score(y_test, test_pred):.4f}")
print(f"Recall: {recall_score(y_test, test_pred):.4f}")
print(f"F1-Score: {f1_score(y_test, test_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, test_pred))

# Individual model evaluation
print("\nIndividual Model Performance:")
models = {
    "CNN": (lambda X: (cnn_model.predict(X) > 0.5).astype(int).flatten(),
X_test_pad),
    "Logistic Regression": (log_clf.predict, X_test_tfidf),
    "Naive Bayes": (nb_clf.predict, X_test_tfidf),
    "SVM": (svm_clf.predict, X_test_tfidf)
}

for name, (predict_fn, X) in models.items():
    y_pred = predict_fn(X)
    print(f"\n{name}:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred):.4f}")
    print(f"Recall: {recall_score(y_test, y_pred):.4f}")
    print(f"F1-Score: {f1_score(y_test, y_pred):.4f}")

# Manual testing
try:
    test_df = pd.read_csv('manual_testing.csv')
    test_df['cleaned_text'] = test_df['text'].apply(clean_text)

    # Process text
    test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
    test_pad = pad_sequences(test_seq, maxlen=max_len)
    test_tfidf = tfidf.transform(test_df['cleaned_text'])

    # Predict
    test_pred = ensemble_predict(test_tfidf, test_pad)
    test_df['prediction'] = ['Fake' if p == 0 else 'True' for p in test_pred]
    print("\nManual Testing Results:")
    print(test_df[['text', 'prediction']].head())
except Exception as e:
    print(f"\nManual testing error: {str(e)}")
# Installation (run this first)
!pip install tensorflow scikit-learn numpy pandas nltk
```

```
!python -m pip install --upgrade pip
nltk.download('stopwords')
nltk.download('wordnet')

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Dropout,
Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from sklearn.base import BaseEstimator, ClassifierMixin, TransformerMixin
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from scipy.sparse import hstack
from tensorflow.keras.regularizers import l2

# Load and preprocess data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1  # 1 for true news
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text cleaning
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
```

```
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df['cleaned_text'], df['label'], test_size=0.2, random_state=42
)

# Text vectorization for LSTM
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# Keras Classifier Wrapper for Voting
class KerasVotingWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, build_fn, epochs=5, batch_size=32):
        self.build_fn = build_fn
        self.epochs = epochs
        self.batch_size = batch_size
        self.model = None

    def fit(self, X, y):
        self.model = self.build_fn()
        self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size,
verbose=1)
        return self

    def predict(self, X):
        return (self.model.predict(X) > 0.5).astype(int).flatten()

    def predict_proba(self, X):
        proba = self.model.predict(X)
        return np.hstack([1-proba, proba])  # Return probabilities for both
classes
```

```
    def get_params(self, deep=True):
        return {'build_fn': self.build_fn,
                'epochs': self.epochs,
                'batch_size': self.batch_size}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

# LSTM Model
def build_lstm_model():
    input_layer = Input(shape=(max_len,))
    embedding = Embedding(max_words, 128, input_length=max_len)(input_layer)

    # Bidirectional LSTM with dropout
    lstm = Bidirectional(LSTM(64, return_sequences=True, dropout=0.2,
recurrent_dropout=0.2))(embedding)
    lstm = Bidirectional(LSTM(32, dropout=0.2, recurrent_dropout=0.2))(lstm)

    dense = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(lstm)
    dropout = Dropout(0.5)(dense)
    output = Dense(1, activation='sigmoid')(dropout)

    model = Model(inputs=input_layer, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# First train the LSTM model
print("Step 1: Training LSTM model...")
lstm_model = KerasVotingWrapper(build_fn=build_lstm_model, epochs=5,
batch_size=32)
lstm_model.fit(X_train_pad, y_train)

# TF-IDF Vectorizer for traditional models
print("Step 2: Creating TF-IDF features...")
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# Traditional ML models
print("Training traditional models...")
svm_model = SVC(probability=True, kernel='linear', random_state=42)
```

```
nb_model = MultinomialNB(alpha=0.1)
lr_model = LogisticRegression(random_state=42, max_iter=1000)

# We need to create a pipeline for traditional models to handle text input
def get_traditional_model_pipeline(model):
    return Pipeline([
        ('tfidf', TfidfVectorizer(max_features=5000, ngram_range=(1, 2))),
        ('model', model)
    ])

# Voting Classifier
print("Step 3: Creating Voting Classifier...")

# We need separate pipelines for the voting classifier:
# 1. LSTM model that takes sequence input
# 2. Traditional models that take raw text input

class LSTMTransformer(TransformerMixin):
    def __init__(self, lstm_model, tokenizer, max_len):
        self.lstm_model = lstm_model
        self.tokenizer = tokenizer
        self.max_len = max_len

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        sequences = self.tokenizer.texts_to_sequences(X)
        padded = pad_sequences(sequences, maxlen=self.max_len)
        return (self.lstm_model.predict_proba(padded)[:, 1] >
0.5).astype(int)

# Create a voting classifier with different input types
class HeterogeneousVotingClassifier(VotingClassifier):
    def __init__(self, estimators, voting='soft'):
        super().__init__(estimators, voting=voting)

    def fit(self, X, y):
        # Fit each estimator with its appropriate input
        for name, estimator in self.estimators:
            if 'lstm' in name:
                # For LSTM, we need to preprocess the text
                X_transformed = pad_sequences(
                    self.estimators_[0][1].steps[0][1].tokenizer.texts_to_seq
uences(X),
```

```
                    maxlen=self.estimators_[0][1].steps[0][1].max_len
                )
                estimator.steps[1][1].fit(X_transformed, y)
            else:
                # For traditional models, use TF-IDF
                estimator.fit(X, y)
        return self

    def predict(self, X):
        # Collect predictions from all estimators
        predictions = []
        for name, estimator in self.estimators:
            if 'lstm' in name:
                X_transformed = pad_sequences(
                    self.estimators_[0][1].steps[0][1].tokenizer.texts_to_seq
uences(X),
                    maxlen=self.estimators_[0][1].steps[0][1].max_len
                )
                pred = estimator.steps[1][1].predict(X_transformed)
            else:
                pred = estimator.predict(X)
            predictions.append(pred)
        return self._predict_vote(predictions)

# Create pipelines for each model
lstm_pipeline = Pipeline([
    ('preprocess', LSTMTransformer(lstm_model, tokenizer, max_len)),
    ('lstm', lstm_model)
])

svm_pipeline = get_traditional_model_pipeline(svm_model)
nb_pipeline = get_traditional_model_pipeline(nb_model)
lr_pipeline = get_traditional_model_pipeline(lr_model)

# Create voting classifier
voting_model = HeterogeneousVotingClassifier(
    estimators=[
        ('lstm', lstm_pipeline),
        ('svm', svm_pipeline),
        ('nb', nb_pipeline),
        ('lr', lr_pipeline)
    ],
    voting='soft'
)
```

```
# Fit the voting classifier
print("Training Voting Classifier...")
voting_model.fit(X_train, y_train)

# Evaluation function
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Evaluate LSTM alone
print("\nLSTM Model Performance:")
lstm_pred = lstm_model.predict(X_test_pad)
print_metrics(y_test, lstm_pred, "LSTM Alone")

# Evaluate Voting Classifier
print("\nVoting Classifier Performance:")
voting_pred = voting_model.predict(X_test)
print_metrics(y_test, voting_pred, "Voting Classifier")

# Manual testing with metrics
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # Predict
        test_pred = voting_model.predict(test_df['cleaned_text'])
        test_df['prediction'] = test_pred

        # If manual testing has labels, calculate metrics
        if 'label' in test_df.columns:
            print("\nManual Testing Metrics:")
            print_metrics(test_df['label'], test_pred, "Manual Testing")
        else:
            print("\nManual Testing Results (no labels available):")

        print(test_df[['text', 'prediction']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
```

```
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
except Exception as e:
    print(f"\nError during manual testing: {str(e)}")

# Individual model performance
print("\nIndividual Model Performance on Test Set:")
for name, estimator in voting_model.estimators:
    if 'lstm' in name:
        # For LSTM, we need to preprocess the text
        X_transformed = pad_sequences(
            tokenizer.texts_to_sequences(X_test),
            maxlen=max_len
        )
        model_pred = estimator.steps[1][1].predict(X_transformed)
    else:
        model_pred = estimator.predict(X_test)

    print(f"\n{name}:")
    print(f"Accuracy:  {accuracy_score(y_test, model_pred):.4f}")
    print(f"Precision: {precision_score(y_test, model_pred):.4f}")
    print(f"Recall:    {recall_score(y_test, model_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_test, model_pred):.4f}")
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import roc_curve, auc
from PIL import Image

# Load the dataset (Replace with actual file path)
df = pd.read_csv("model_results.csv")  # Ensure this contains 'Model' and
'Accuracy'

# Simulate ROC-AUC values (Estimated as Accuracy - 0.02 for variation)
df["ROC_AUC"] = df["Accuracy"] - 0.02  # Adjust this logic as needed

# Sorting models by Accuracy
df = df.sort_values(by="Accuracy", ascending=False)

# Plot Accuracy Bar Chart
plt.figure(figsize=(12, 6))
sns.barplot(x=df["Accuracy"], y=df["Model"], palette="Blues_r")
plt.xlabel("Accuracy (%)")
```

```
plt.ylabel("Models")
plt.title("Model Accuracy Comparison")
plt.xlim([min(df["Accuracy"]) - 0.5, 100])
plt.grid(axis="x", linestyle="--", alpha=0.6)
plt.savefig("accuracy_plot.png", bbox_inches="tight")

# Simulated ROC-AUC Graphs
plt.figure(figsize=(10, 8))
colors = sns.color_palette("husl", len(df))

for i, row in df.iterrows():
    fpr = np.linspace(0, 1, 100)  # Simulated False Positive Rates
    tpr = np.sqrt(fpr) * row["ROC_AUC"]  # Simulated True Positive Rates
(structured estimation)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, color=colors[i], lw=2, label=f'{row["Model"]} (AUC =
{roc_auc:.2f})')

plt.plot([0, 1], [0, 1], linestyle="--", color="gray", lw=2)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Simulated ROC-AUC Curves for Models")
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.savefig("roc_auc_plot.png", bbox_inches="tight")

# Combine both plots into one image
acc_img = Image.open("accuracy_plot.png")
roc_img = Image.open("roc_auc_plot.png")

combined = Image.new("RGB", (max(acc_img.width, roc_img.width),
acc_img.height + roc_img.height))
combined.paste(acc_img, (0, 0))
combined.paste(roc_img, (0, acc_img.height))
combined.save("model_comparison.png")

plt.show()

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset (replace with actual filename)
df = pd.read_csv("model_results.csv")  # Ensure the file contains 'Model' and
'Accuracy'
```

```
# Normalize column names to avoid errors
df.columns = df.columns.str.strip().str.lower()

# Ensure correct column names and rename if needed
df.rename(columns={"accuracy": "Accuracy", "model": "Model"}, inplace=True)

# Sorting models by Accuracy (ascending for better visualization in line
plot)
df = df.sort_values(by="Accuracy")

# Plot Accuracy Line Chart
plt.figure(figsize=(12, 6))
sns.lineplot(x=df["Model"], y=df["Accuracy"], marker="o", linestyle="-",
color="b")

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha="right", fontsize=10)

# Formatting
plt.xlabel("Models", fontsize=14)
plt.ylabel("Accuracy (%)", fontsize=14)
plt.title("Model Accuracy Trend", fontsize=16)
plt.ylim([max(0, min(df["Accuracy"]) - 5), 1])  # Dynamic Y-axis scaling
plt.grid(True, linestyle="--", alpha=0.5)

# Save and Show Plot
plt.savefig("accuracy_line_plot.png", bbox_inches="tight")
plt.show()

# Installation (run this first)
!pip install tensorflow scikit-learn numpy pandas nltk
!python -m pip install --upgrade pip
nltk.download('stopwords')
nltk.download('wordnet')

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Conv1D,
GlobalMaxPooling1D, Dense, Dropout, Concatenate
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.ensemble import StackingClassifier
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from sklearn.base import BaseEstimator, ClassifierMixin, TransformerMixin
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline, FeatureUnion
from scipy.sparse import hstack

# Load and preprocess data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1  # 1 for true news
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text cleaning
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df['cleaned_text'], df['label'], test_size=0.2, random_state=42
)

# Text vectorization for CNN
max_words = 10000
max_len = 200
```

```
tokenizer = Tokenizer(num_words=max_words)
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)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# Proper Keras Classifier Wrapper for Stacking
class KerasStackingWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, build_fn, epochs=3, batch_size=32):
        self.build_fn = build_fn
        self.epochs = epochs
        self.batch_size = batch_size
        self.model = None

    def fit(self, X, y):
        self.model = self.build_fn()
        self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size,
verbose=0)
        return self

    def predict(self, X):
        return (self.model.predict(X) > 0.5).astype(int).flatten()

    def predict_proba(self, X):
        proba = self.model.predict(X)
        return np.hstack([1-proba, proba])  # Return probabilities for both
classes

    def get_params(self, deep=True):
        return {'build_fn': self.build_fn,
                'epochs': self.epochs,
                'batch_size': self.batch_size}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

# CNN Model
def build_cnn_model():
    input_layer = Input(shape=(max_len,))
```

```
    embedding = Embedding(max_words, 128)(input_layer)

    conv1 = Conv1D(128, 3, activation='relu')(embedding)
    pool1 = GlobalMaxPooling1D()(conv1)

    conv2 = Conv1D(128, 4, activation='relu')(embedding)
    pool2 = GlobalMaxPooling1D()(conv2)

    conv3 = Conv1D(128, 5, activation='relu')(embedding)
    pool3 = GlobalMaxPooling1D()(conv3)

    merged = Concatenate()([pool1, pool2, pool3])

    dense = Dense(64, activation='relu')(merged)
    dropout = Dropout(0.5)(dense)
    output = Dense(1, activation='sigmoid')(dropout)

    model = Model(inputs=input_layer, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# First train the CNN model
print("Step 1: Training CNN model...")
cnn_model = KerasStackingWrapper(build_fn=build_cnn_model, epochs=3,
batch_size=32)
cnn_model.fit(X_train_pad, y_train)

# Create feature extractor from CNN
class CNNFeatureExtractor(TransformerMixin):
    def __init__(self, cnn_model):
        self.cnn_model = cnn_model
        self.feature_model = None

    def fit(self, X, y=None):
        # Create a model that outputs the penultimate layer
        self.feature_model = Model(inputs=self.cnn_model.model.input,
                                   outputs=self.cnn_model.model.layers[-
2].output)
        return self

    def transform(self, X):
        return self.feature_model.predict(X)

# TF-IDF Vectorizer
```

```
print("Step 2: Creating TF-IDF features...")
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# CNN Features
print("Creating CNN features...")
cnn_feature_extractor = CNNFeatureExtractor(cnn_model)
cnn_feature_extractor.fit(X_train_pad)  # Just to initialize the feature
model
X_train_cnn = cnn_feature_extractor.transform(X_train_pad)
X_test_cnn = cnn_feature_extractor.transform(X_test_pad)

# Combine features
print("Combining features...")
X_train_features = hstack([X_train_tfidf, X_train_cnn])
X_test_features = hstack([X_test_tfidf, X_test_cnn])

# Stacking Classifier
print("Step 3: Training Stacking Classifier...")
svm_model = SVC(probability=True, kernel='linear', random_state=42)
nb_model = MultinomialNB()
lr_model = LogisticRegression(random_state=42)

stacking_model = StackingClassifier(
    estimators=[
        ('svm', svm_model),
        ('nb', nb_model),
        ('lr', lr_model)
    ],
    final_estimator=LogisticRegression(),
    stack_method='auto',
    n_jobs=-1,
    passthrough=True
)

stacking_model.fit(X_train_features, y_train)

# Evaluation function
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
```

```
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Evaluate on test set
test_pred = stacking_model.predict(X_test_features)
print_metrics(y_test, test_pred, "Test Set")

# Manual testing with metrics
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # Transform test data
        test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        test_pad = pad_sequences(test_seq, maxlen=max_len)

        test_tfidf = tfidf.transform(test_df['cleaned_text'])
        test_cnn = cnn_feature_extractor.transform(test_pad)
        test_features = hstack([test_tfidf, test_cnn])

        # Predict
        test_pred = stacking_model.predict(test_features)
        test_df['prediction'] = test_pred

        # If manual testing has labels, calculate metrics
        if 'label' in test_df.columns:
            print("\nManual Testing Metrics:")
            print_metrics(test_df['label'], test_pred, "Manual Testing")
        else:
            print("\nManual Testing Results (no labels available):")

        print(test_df[['text', 'prediction']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
except Exception as e:
    print(f"\nError during manual testing: {str(e)}")

# Individual model performance
print("\nIndividual Model Performance on Test Set:")
for name, model in stacking_model.named_estimators_.items():
```

```
    model_pred = model.predict(X_test_features)
    print(f"\n{name}:")
    print(f"Accuracy:  {accuracy_score(y_test, model_pred):.4f}")
    print(f"Precision: {precision_score(y_test, model_pred):.4f}")
    print(f"Recall:    {recall_score(y_test, model_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_test, model_pred):.4f}")
# Installation (run this first)
!pip install tensorflow scikit-learn numpy pandas nltk
!python -m pip install --upgrade pip
nltk.download('stopwords')
nltk.download('wordnet')

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Conv1D,
GlobalMaxPooling1D, Dense, Dropout, Concatenate
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import BaseEstimator, ClassifierMixin

# Load and preprocess data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0
true_df['label'] = 1
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text cleaning
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()
```

```
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df['cleaned_text'], df['label'], test_size=0.2, random_state=42
)

# Text vectorization
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# Proper Keras Classifier Wrapper
class KerasClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, build_fn, epochs=3, batch_size=32):
        self.build_fn = build_fn
        self.epochs = epochs
        self.batch_size = batch_size
        self.model = None

    def fit(self, X, y):
        self.model = self.build_fn()
        self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size,
verbose=0)
        return self

    def predict(self, X):
        return (self.model.predict(X) > 0.5).astype(int).flatten()

    def predict_proba(self, X):
        proba = self.model.predict(X)
        return np.hstack([1-proba, proba])
```

```
    def get_params(self, deep=True):
        return {'build_fn': self.build_fn,
                'epochs': self.epochs,
                'batch_size': self.batch_size}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

# CNN Model
def build_cnn_model():
    input_layer = Input(shape=(max_len,))
    embedding = Embedding(max_words, 128)(input_layer)

    conv1 = Conv1D(128, 3, activation='relu')(embedding)
    pool1 = GlobalMaxPooling1D()(conv1)

    conv2 = Conv1D(128, 4, activation='relu')(embedding)
    pool2 = GlobalMaxPooling1D()(conv2)

    conv3 = Conv1D(128, 5, activation='relu')(embedding)
    pool3 = GlobalMaxPooling1D()(conv3)

    merged = Concatenate()([pool1, pool2, pool3])

    dense = Dense(64, activation='relu')(merged)
    dropout = Dropout(0.5)(dense)
    output = Dense(1, activation='sigmoid')(dropout)

    model = Model(inputs=input_layer, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# TF-IDF Vectorizer
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# Initialize models
cnn_model = KerasClassifier(build_fn=build_cnn_model, epochs=3,
batch_size=32)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
# Train CNN first to get features
print("Training CNN model...")
cnn_model.fit(X_train_pad, y_train)

# Get CNN predictions as features
X_train_cnn_pred = cnn_model.predict_proba(X_train_pad)
X_test_cnn_pred = cnn_model.predict_proba(X_test_pad)

# Combine features
X_train_combined = np.hstack([X_train_tfidf.toarray(), X_train_cnn_pred])
X_test_combined = np.hstack([X_test_tfidf.toarray(), X_test_cnn_pred])

# Train Random Forest
print("\nTraining Random Forest on combined features...")
rf_model.fit(X_train_combined, y_train)

# Evaluate
test_pred = rf_model.predict(X_test_combined)
print("\nTest Accuracy:", accuracy_score(y_test, test_pred))
print("\nClassification Report:")
print(classification_report(y_test, test_pred))

# Feature importance
print("\nFeature Importances (Top 20):")
importances = rf_model.feature_importances_
top_indices = np.argsort(importances)[-20:][::-1]
feature_names = [f"TF-IDF_{i}" for i in range(X_train_tfidf.shape[1])] + \
                [f"CNN_Proba_{i}" for i in range(X_train_cnn_pred.shape[1])]
for idx in top_indices:
    print(f"{feature_names[idx]}: {importances[idx]:.4f}")

# Manual testing
try:
    test_df = pd.read_csv('manual_testing.csv')
    test_df['cleaned_text'] = test_df['text'].apply(clean_text)

    # Process text
    test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
    test_pad = pad_sequences(test_seq, maxlen=max_len)
    test_tfidf = tfidf.transform(test_df['cleaned_text'])
    test_cnn_pred = cnn_model.predict_proba(test_pad)
    test_combined = np.hstack([test_tfidf.toarray(), test_cnn_pred])

    # Predict
```

```
    test_pred = rf_model.predict(test_combined)
    test_df['prediction'] = ['Fake' if p == 0 else 'True' for p in test_pred]
    print("\nManual Testing Results:")
    print(test_df[['text', 'prediction']].head())
except Exception as e:
    print(f"\nManual testing error: {str(e)}")
# Installation (run this first)
!pip install tensorflow scikit-learn numpy pandas nltk
!python -m pip install --upgrade pip
nltk.download('stopwords')
nltk.download('wordnet')

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Dropout,
Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from sklearn.base import BaseEstimator, ClassifierMixin, TransformerMixin
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from scipy.sparse import hstack
from tensorflow.keras.regularizers import l2

# Load and preprocess data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1  # 1 for true news
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text cleaning
```

```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df['cleaned_text'], df['label'], test_size=0.2, random_state=42
)

# Text vectorization for LSTM
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# Keras Classifier Wrapper for Stacking
class KerasStackingWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, build_fn, epochs=5, batch_size=32):
        self.build_fn = build_fn
        self.epochs = epochs
        self.batch_size = batch_size
        self.model = None

    def fit(self, X, y):
        self.model = self.build_fn()
        self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size,
verbose=1)
        return self
```

```
    def predict(self, X):
        return (self.model.predict(X) > 0.5).astype(int).flatten()

    def predict_proba(self, X):
        proba = self.model.predict(X)
        return np.hstack([1-proba, proba])  # Return probabilities for both
classes

    def get_params(self, deep=True):
        return {'build_fn': self.build_fn,
                'epochs': self.epochs,
                'batch_size': self.batch_size}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

# LSTM Model
def build_lstm_model():
    input_layer = Input(shape=(max_len,))
    embedding = Embedding(max_words, 128, input_length=max_len)(input_layer)

    # Bidirectional LSTM with dropout
    lstm = Bidirectional(LSTM(64, return_sequences=True, dropout=0.2,
recurrent_dropout=0.2))(embedding)
    lstm = Bidirectional(LSTM(32, dropout=0.2, recurrent_dropout=0.2))(lstm)

    dense = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(lstm)
    dropout = Dropout(0.5)(dense)
    output = Dense(1, activation='sigmoid')(dropout)

    model = Model(inputs=input_layer, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# First train the LSTM model
print("Step 1: Training LSTM model...")
lstm_model = KerasStackingWrapper(build_fn=build_lstm_model, epochs=5,
batch_size=32)
lstm_model.fit(X_train_pad, y_train)

# Create feature extractor from LSTM
```

```
class LSTMFeatureExtractor(TransformerMixin):
    def __init__(self, lstm_model):
        self.lstm_model = lstm_model
        self.feature_model = None

    def fit(self, X, y=None):
        # Create a model that outputs the penultimate layer
        self.feature_model = Model(inputs=self.lstm_model.model.input,
                                   outputs=self.lstm_model.model.layers[-
2].output)
        return self

    def transform(self, X):
        return self.feature_model.predict(X)

# TF-IDF Vectorizer
print("Step 2: Creating TF-IDF features...")
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# LSTM Features
print("Creating LSTM features...")
lstm_feature_extractor = LSTMFeatureExtractor(lstm_model)
lstm_feature_extractor.fit(X_train_pad)  # Just to initialize the feature
model
X_train_lstm = lstm_feature_extractor.transform(X_train_pad)
X_test_lstm = lstm_feature_extractor.transform(X_test_pad)

# Combine features
print("Combining features...")
X_train_features = hstack([X_train_tfidf, X_train_lstm])
X_test_features = hstack([X_test_tfidf, X_test_lstm])

# Stacking Classifier
print("Step 3: Training Stacking Classifier...")
svm_model = SVC(probability=True, kernel='linear', random_state=42)
nb_model = MultinomialNB(alpha=0.1)
lr_model = LogisticRegression(random_state=42, max_iter=1000)

stacking_model = StackingClassifier(
    estimators=[
        ('svm', svm_model),
        ('nb', nb_model),
        ('lr', lr_model)
```

```
    ],
    final_estimator=LogisticRegression(),
    stack_method='auto',
    n_jobs=-1,
    passthrough=True
)

stacking_model.fit(X_train_features, y_train)

# Evaluation function
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# Evaluate LSTM alone
print("\nLSTM Model Performance:")
lstm_pred = lstm_model.predict(X_test_pad)
print_metrics(y_test, lstm_pred, "LSTM Alone")

# Evaluate on test set
print("\nStacking Ensemble Performance:")
test_pred = stacking_model.predict(X_test_features)
print_metrics(y_test, test_pred, "Stacking Ensemble")

# Manual testing with metrics
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # Transform test data
        test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        test_pad = pad_sequences(test_seq, maxlen=max_len)

        test_tfidf = tfidf.transform(test_df['cleaned_text'])
        test_lstm = lstm_feature_extractor.transform(test_pad)
        test_features = hstack([test_tfidf, test_lstm])

        # Predict
```

```
        test_pred = stacking_model.predict(test_features)
        test_df['prediction'] = test_pred

        # If manual testing has labels, calculate metrics
        if 'label' in test_df.columns:
            print("\nManual Testing Metrics:")
            print_metrics(test_df['label'], test_pred, "Manual Testing")
        else:
            print("\nManual Testing Results (no labels available):")

        print(test_df[['text', 'prediction']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
except Exception as e:
    print(f"\nError during manual testing: {str(e)}")

# Individual model performance
print("\nIndividual Model Performance on Test Set:")
for name, model in stacking_model.named_estimators_.items():
    model_pred = model.predict(X_test_features)
    print(f"\n{name}:")
    print(f"Accuracy:  {accuracy_score(y_test, model_pred):.4f}")
    print(f"Precision: {precision_score(y_test, model_pred):.4f}")
    print(f"Recall:    {recall_score(y_test, model_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_test, model_pred):.4f}")
# Installation (run this first)
!pip install tensorflow scikit-learn numpy pandas nltk
!python -m pip install --upgrade pip
nltk.download('stopwords')
nltk.download('wordnet')

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Dropout,
Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from sklearn.base import BaseEstimator, ClassifierMixin, TransformerMixin
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from scipy.sparse import hstack
from tensorflow.keras.regularizers import l2

# Load and preprocess data
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

fake_df['label'] = 0  # 0 for fake news
true_df['label'] = 1  # 1 for true news
df = pd.concat([fake_df, true_df]).sample(frac=1).reset_index(drop=True)

# Text cleaning
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if
word not in stop_words])
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df['cleaned_text'], df['label'], test_size=0.2, random_state=42
)

# Text vectorization for LSTM
max_words = 10000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
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)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# LSTM Model
def build_lstm_model():
    input_layer = Input(shape=(max_len,))
    embedding = Embedding(max_words, 128, input_length=max_len)(input_layer)

    # Bidirectional LSTM with dropout
    lstm = Bidirectional(LSTM(64, return_sequences=True, dropout=0.2,
recurrent_dropout=0.2))(embedding)
    lstm = Bidirectional(LSTM(32, dropout=0.2, recurrent_dropout=0.2))(lstm)

    dense = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(lstm)
    dropout = Dropout(0.5)(dense)
    output = Dense(1, activation='sigmoid')(dropout)

    model = Model(inputs=input_layer, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# Train the LSTM model
print("Step 1: Training LSTM model...")
lstm_model = build_lstm_model()
lstm_model.fit(X_train_pad, y_train, epochs=5, batch_size=32,
validation_split=0.1)

# Create feature extractor from LSTM
class LSTMFeatureExtractor(TransformerMixin):
    def __init__(self, lstm_model):
        self.lstm_model = lstm_model
        self.feature_model = None

    def fit(self, X, y=None):
        # Create a model that outputs the penultimate layer
        self.feature_model = Model(inputs=self.lstm_model.input,
                                   outputs=self.lstm_model.layers[-2].output)
        return self

    def transform(self, X):
```

```
        return self.feature_model.predict(X)

# TF-IDF Vectorizer
print("Step 2: Creating TF-IDF features...")
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# LSTM Features
print("Creating LSTM features...")
lstm_feature_extractor = LSTMFeatureExtractor(lstm_model)
lstm_feature_extractor.fit(X_train_pad)  # Initialize the feature model
X_train_lstm = lstm_feature_extractor.transform(X_train_pad)
X_test_lstm = lstm_feature_extractor.transform(X_test_pad)

# Combine features
print("Combining features...")
X_train_combined = hstack([X_train_tfidf, X_train_lstm])
X_test_combined = hstack([X_test_tfidf, X_test_lstm])

# Random Forest Classifier
print("Step 3: Training Random Forest on combined features...")
rf_model = RandomForestClassifier(
    n_estimators=200,
    max_depth=50,
    min_samples_split=5,
    min_samples_leaf=2,
    max_features='sqrt',
    random_state=42,
    n_jobs=-1
)

rf_model.fit(X_train_combined, y_train)

# Evaluation function
def print_metrics(y_true, y_pred, label="Test"):
    print(f"\n{label} Metrics:")
    print("----------------------")
    print(f"Accuracy:  {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall:    {recall_score(y_true, y_pred):.4f}")
    print(f"F1-Score:  {f1_score(y_true, y_pred):.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))
```

```
# Evaluate LSTM alone
print("\nLSTM Model Performance:")
lstm_pred = (lstm_model.predict(X_test_pad) > 0.5).astype(int)
print_metrics(y_test, lstm_pred.flatten(), "LSTM Alone")

# Evaluate Random Forest on combined features
print("\nRandom Forest with Combined Features Performance:")
rf_pred = rf_model.predict(X_test_combined)
print_metrics(y_test, rf_pred, "Random Forest")

# Feature importance analysis (for TF-IDF features only)
print("\nTop 20 Important TF-IDF Features:")
feature_names = tfidf.get_feature_names_out()
importances = rf_model.feature_importances_[:len(feature_names)]  # Get
importances for TF-IDF features only
indices = np.argsort(importances)[-20:]  # Top 20 important features

for i in indices:
    print(f"{feature_names[i]}: {importances[i]:.4f}")

# Manual testing with metrics
try:
    test_df = pd.read_csv('manual_testing.csv')
    if 'text' in test_df.columns:
        test_df['cleaned_text'] = test_df['text'].apply(clean_text)

        # Transform test data
        test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])
        test_pad = pad_sequences(test_seq, maxlen=max_len)

        test_tfidf = tfidf.transform(test_df['cleaned_text'])
        test_lstm = lstm_feature_extractor.transform(test_pad)
        test_combined = hstack([test_tfidf, test_lstm])

        # Predict
        test_pred = rf_model.predict(test_combined)
        test_df['prediction'] = test_pred

        # If manual testing has labels, calculate metrics
        if 'label' in test_df.columns:
            print("\nManual Testing Metrics:")
            print_metrics(test_df['label'], test_pred, "Manual Testing")
        else:
            print("\nManual Testing Results (no labels available):")
```

```
        print(test_df[['text', 'prediction']].head())
    else:
        print("\nmanual_testing.csv doesn't contain 'text' column")
except FileNotFoundError:
    print("\nmanual_testing.csv not found - skipping manual test
predictions")
except Exception as e:
    print(f"\nError during manual testing: {str(e)}")
```