```
# ClassiNews: Intelligent News Article Categorization System
# Enhanced Version with All Features

# ============================================================
# STEP 1: Import Libraries
# ============================================================
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from wordcloud import WordCloud
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_recall_fscore_support
import warnings
warnings.filterwarnings('ignore')
import gradio as gr
import re
import kagglehub
from kagglehub import KaggleDatasetAdapter
import joblib
import pickle
from datetime import datetime

# Download NLTK data
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)

print("All libraries imported successfully!")

# ============================================================
# STEP 2: Load Dataset from Kaggle
# ============================================================
print("\nLoading AG News dataset from Kaggle...")

# Load both train and test datasets
train_df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "amananandrai/ag-news-classification-dataset",
    "train.csv"
)

test_df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "amananandrai/ag-news-classification-dataset",
    "test.csv"
)

print(f"Datasets loaded successfully!")
print(f"Train dataset shape: {train_df.shape}")
```

```
print(f"Test dataset shape: {test_df.shape}")

# Check column names
print(f"\nTrain columns: {train_df.columns.tolist()}")
print(f"Test columns: {test_df.columns.tolist()}")

# ============================================================
# STEP 3: Enhanced Data Preprocessing
# ============================================================
print("\n Preprocessing data...")

# Map class indices to category names
category_map = {
    1: 'World',
    2: 'Sports',
    3: 'Business',
    4: 'Science/Tech'
}

# Process train data
train_df['Category'] = train_df['Class Index'].map(category_map)
train_df['Text'] = train_df['Title'] + ' ' + train_df['Description']

# Process test data
test_df['Category'] = test_df['Class Index'].map(category_map)
test_df['Text'] = test_df['Title'] + ' ' + test_df['Description']

# Enhanced text preprocessing function
def preprocess_text(text):
    if pd.isna(text):
        return ""
    # Convert to lowercase
    text = str(text).lower()
    # Remove special characters and digits
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', '', text)
    # Remove extra whitespace
    text = ' '.join(text.split())
    return text

# Apply preprocessing
train_df['Processed_Text'] = train_df['Text'].apply(preprocess_text)
test_df['Processed_Text'] = test_df['Text'].apply(preprocess_text)

# Calculate text length statistics
train_df['Text_Length'] = train_df['Processed_Text'].apply(len)
train_df['Word_Count'] = train_df['Processed_Text'].apply(lambda x: len(x.split()))

test_df['Text_Length'] = test_df['Processed_Text'].apply(len)
test_df['Word_Count'] = test_df['Processed_Text'].apply(lambda x: len(x.split()))

print("Data preprocessing completed!")
print(f"\nTrain dataset sample:")
print(train_df[['Category', 'Text_Length', 'Word_Count']].head())

# Display category distribution
print(f"\nCategory Distribution (Train):")
```

```
print(train_df['Category'].value_counts())
print(f"\nCategory Distribution (Test):")
print(test_df['Category'].value_counts())

# ============================================================
# STEP 4: Enhanced Visualizations
# ============================================================
print("\nCreating visualizations...")

# 1. Category Distribution Comparison
fig1 = make_subplots(
    rows=1, cols=2,
    subplot_titles=('Train Set Distribution', 'Test Set Distribution'),
    specs=[[{'type': 'pie'}, {'type': 'pie'}]]
)

# Train distribution
train_counts = train_df['Category'].value_counts()
fig1.add_trace(
    go.Pie(labels=train_counts.index, values=train_counts.values,
           name="Train", hole=0.3, textinfo='percent+label'),
    row=1, col=1
)

# Test distribution
test_counts = test_df['Category'].value_counts()
fig1.add_trace(
    go.Pie(labels=test_counts.index, values=test_counts.values,
           name="Test", hole=0.3, textinfo='percent+label'),
    row=1, col=2
)

fig1.update_layout(
    height=400,
    title_text="Dataset Category Distribution Comparison",
    showlegend=True
)
fig1.show()

# 2. Text Length Analysis
fig2 = make_subplots(
    rows=1, cols=2,
    subplot_titles=('Train Set Text Length', 'Test Set Text Length'),
    specs=[[{'type': 'box'}, {'type': 'box'}]]
)

for idx, category in enumerate(train_df['Category'].unique()):
    fig2.add_trace(
        go.Box(y=train_df[train_df['Category']==category]['Text_Length'],
               name=category, showlegend=False),
        row=1, col=1
    )

for idx, category in enumerate(test_df['Category'].unique()):
    fig2.add_trace(
        go.Box(y=test_df[test_df['Category']==category]['Text_Length'],
               name=category, showlegend=False),
```

```
        row=1, col=2
    )

fig2.update_layout(
    height=500,
    title_text="Text Length Distribution by Category",
    yaxis_title="Text Length (characters)"
)
fig2.show()

# 3. Word Clouds for Train Set
print("\n☘  Generating Word Clouds for train set categories...")

fig3, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.flatten()

for idx, category in enumerate(train_df['Category'].unique()):
    text = ' '.join(train_df[train_df['Category']==category]['Processed_Text'])
    wordcloud = WordCloud(width=400, height=200,
                          background_color='white',
                          max_words=100).generate(text)

    axes[idx].imshow(wordcloud, interpolation='bilinear')
    axes[idx].set_title(f'{category} News', fontsize=14, fontweight='bold')
    axes[idx].axis('off')

plt.tight_layout()
plt.show()

# ============================================================
# STEP 5: Split Dataset & Vectorization
# ============================================================
print("\nPreparing data for modeling...")

# Use train data for training
X_train = train_df['Processed_Text']
y_train = train_df['Category']

# Use test data for final evaluation
X_test = test_df['Processed_Text']
y_test = test_df['Category']

# TF-IDF Vectorization with better parameters
vectorizer = TfidfVectorizer(
    max_features=10000,
    stop_words='english',
    ngram_range=(1, 2),  # Include bigrams
    min_df=2,
    max_df=0.95
)

X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

print(f"TF-IDF Matrix shape - Train: {X_train_tfidf.shape}")
print(f"TF-IDF Matrix shape - Test: {X_test_tfidf.shape}")
print(f"Vocabulary size: {len(vectorizer.get_feature_names_out())}")
```

```
print(f"Data preparation completed!")
print(f"Training set: {X_train_tfidf.shape[0]} samples")
print(f"Testing set: {X_test_tfidf.shape[0]} samples")

# ============================================================
# STEP 6: Enhanced Model Training
# ============================================================
print("\n Training Multinomial Naive Bayes model...")

model = MultinomialNB(alpha=0.1)
model.fit(X_train_tfidf, y_train)

print("Model training completed!")

# ============================================================
# STEP 7: Enhanced Model Evaluation with Visualizations
# ============================================================
print("\n Evaluating model performance...")

# Predictions
y_pred = model.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred, average='weighted')

print(f"\n{'='*60}")
print(f" MODEL PERFORMANCE SUMMARY (on Official Test Set)")
print(f"{'='*60}")
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision * 100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")
print(f"F1-Score: {f1 * 100:.2f}%")
print(f"{'='*60}")

# Create performance visualization
metrics_df = pd.DataFrame({
   'Metric': ['Accuracy', 'Precision', 'Recall', 'F1-Score'],
   'Value': [accuracy, precision, recall, f1],
   'Percentage': [f'{accuracy*100:.1f}%', f'{precision*100:.1f}%',
            f'{recall*100:.1f}%', f'{f1*100:.1f}%']
})

fig_metrics = px.bar(metrics_df, x='Metric', y='Value',
              title='Model Performance Metrics on Test Set',
              text='Percentage',
              color='Metric',
              color_discrete_sequence=px.colors.qualitative.Set2)
fig_metrics.update_traces(textposition='outside')
fig_metrics.update_layout(yaxis_range=[0, 1])
fig_metrics.show()

# Confusion Matrix Heatmap
conf_matrix = confusion_matrix(y_test, y_pred)
categories = sorted(y_test.unique())

fig_cm = px.imshow(conf_matrix,
            labels=dict(x="Predicted", y="Actual", color="Count"),
```

```
                x=categories,
                y=categories,
                title="Confusion Matrix Heatmap",
                text_auto=True,
                color_continuous_scale='Blues')
fig_cm.update_layout(width=600, height=600)
fig_cm.show()

# Classification Report Table
print("\n Detailed Classification Report:")
report = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()

fig_report = go.Figure(data=[go.Table(
   header=dict(values=list(report_df.columns),
               fill_color='paleturquoise',
               align='left'),
   cells=dict(values=[report_df.index] + [report_df[col] for col in report_df.columns],
              fill_color='lavender',
              align='left'))
])
fig_report.update_layout(
   title="Classification Report Table",
   width=800,
   height=400
)
fig_report.show()

# ============================================================
# STEP 8: Feature Importance Analysis
# ============================================================
print("\n Analyzing important features...")

# Get feature importance for each category
feature_names = vectorizer.get_feature_names_out()
top_n = 10  # Number of top features to show

fig_importance = make_subplots(
   rows=2, cols=2,
   subplot_titles=[f'Top Features: {cat}' for cat in categories],
   vertical_spacing=0.15,
   horizontal_spacing=0.15
)

for idx, category in enumerate(categories):
   row = idx // 2 + 1
   col = idx % 2 + 1

   # Get feature log probabilities for this category
   feature_log_prob = model.feature_log_prob_[idx]

   # Get top N features
   top_indices = np.argsort(feature_log_prob)[-top_n:]
   top_features = [feature_names[i] for i in top_indices]
   top_scores = feature_log_prob[top_indices]

   fig_importance.add_trace(
```

```
        go.Bar(x=top_scores, y=top_features, orientation='h',
               name=category, marker_color=px.colors.qualitative.Set2[idx]),
        row=row, col=col
    )

    fig_importance.update_xaxes(title_text="Log Probability", row=row, col=col)
    fig_importance.update_yaxes(title_text="Features", row=row, col=col)

fig_importance.update_layout(
    height=800,
    title_text="Top Important Features per Category",
    showlegend=False
)
fig_importance.show()

# ============================================================
# STEP 9: Interactive Web Interface with Gradio (ALL FEATURES)
# ============================================================
print("\n Creating interactive web interface with all features...")

# Prediction history storage
prediction_history = []

def predict_category(text):
    """Predict category for input text"""
    if not text or text.strip() == "":
        return "Please enter some text!"

    # Preprocess
    processed_text = preprocess_text(text)

    # Vectorize
    text_vectorized = vectorizer.transform([processed_text])

    # Predict
    prediction = model.predict(text_vectorized)[0]
    probabilities = model.predict_proba(text_vectorized)[0]

    # Store in history
    prediction_history.append({
        'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        'text': text[:100],
        'prediction': prediction,
        'confidence': max(probabilities) * 100
    })

    # Create result with visual bars
    result = f"*Predicted Category:* {prediction}\n\n"
    result += "*Confidence Scores:*\n"

    # Sort by probability
    sorted_indices = np.argsort(probabilities)[::-1]

    for idx in sorted_indices:
        category = model.classes_[idx]
        prob_percent = probabilities[idx] * 100
```

```
        # Add color based on confidence
        if prob_percent > 70:
            confidence_icon = "🟢"
        elif prob_percent > 40:
            confidence_icon = "🟡"
        else:
            confidence_icon = "🔴"

        # Create visual bar
        bar = "█" * int(prob_percent / 5)
        result += f"{confidence_icon} {category}: {prob_percent:.1f}% {bar}\n"

    return result

def predict_batch(texts):
    """Predict categories for multiple texts"""
    results = []
    for i, text in enumerate(texts.split('\n'), 1):
        if text.strip():
            result = predict_category(text)
            results.append(f"*Article {i}:* {text[:80]}...\n{result}\n{'–'*50}")
    return '\n'.join(results)

def compare_articles(text1, text2):
    """Compare two articles side by side"""
    if not text1 or not text2:
        return "Please enter text for both articles"

    pred1 = predict_category(text1)
    pred2 = predict_category(text2)

    result = f"*Article 1 Classification:*\n{pred1}\n\n"
    result += f"*Article 2 Classification:*\n{pred2}\n\n"

    # Extract categories for comparison
    cat1 = pred1.split('*Predicted Category:*')[1].split('\n')[0].strip()
    cat2 = pred2.split('*Predicted Category:*')[1].split('\n')[0].strip()

    result += "*Comparison:*\n"
    if cat1 == cat2:
        result += f"✅ Both articles belong to the same category: *{cat1}*"
    else:
        result += f"❌ Articles belong to different categories: *{cat1}* vs *{cat2}*"

    return result

def get_prediction_history():
    """Get recent prediction history"""
    if not prediction_history:
        return "No predictions yet"

    result = "*Recent Predictions:*\n\n"
    for i, pred in enumerate(reversed(prediction_history[-10:]), 1):
        result += f"{i}. [{pred['timestamp']}] {pred['text']}...\n"
        result += f"   Category: {pred['prediction']} (Confidence: {pred['confidence']:.1f}%)\n\n"

    return result
```

```
def analyze_text_stats(text):
    """Analyze text statistics along with prediction"""
    if not text or text.strip() == "":
        return "Please enter some text"

    processed = preprocess_text(text)
    words = processed.split()

    stats = f"*Text Analysis:*\n\n"
    stats += f"- Total Characters: {len(text)}\n"
    stats += f"- Total Words: {len(words)}\n"
    stats += f"- Unique Words: {len(set(words))}\n"
    stats += f"- Average Word Length: {np.mean([len(w) for w in words]):.1f}\n"
    stats += f"- Lexical Diversity: {len(set(words))/len(words)*100:.1f}%\n\n"

    stats += predict_category(text)

    return stats

def export_results():
    """Export prediction history to CSV"""
    if not prediction_history:
        return "No predictions to export"

    df = pd.DataFrame(prediction_history)
    filename = f"predictions_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv"
    df.to_csv(filename, index=False)

    return f"✅ Results exported to {filename}"

# Create Gradio interface
demo = gr.Blocks(title="ClassiNews Pro", theme=gr.themes.Soft())

with demo:
    gr.Markdown("# 📰 ClassiNews Pro: Advanced News Categorization System")
    gr.Markdown("### AI-Powered Classification: World | Sports | Business | Science/Tech")

    with gr.Tabs():
        with gr.TabItem("🔍 Quick Classify"):
            with gr.Row():
                with gr.Column(scale=2):
                    input_text = gr.Textbox(
                        label="Enter news article text",
                        placeholder="Paste your news article here...",
                        lines=6
                    )

                    with gr.Row():
                        classify_btn = gr.Button("🚀 Classify", variant="primary", size="lg")
                        analyze_btn = gr.Button("📊 Analyze Stats", variant="secondary")

                    gr.Markdown("### Quick Examples")
                    example_btns = gr.Examples(
                        examples=[
                            ["The stock market surged today as tech companies reported record profits and
investor confidence reached new heights."],
```

```
                ["Manchester United wins Premier League championship in dramatic final match against Liverpool at Old Trafford."],
                ["NASA scientists discover water on Mars, raising possibilities of ancient microbial life on the red planet."],
                ["United Nations Security Council convenes emergency session to address escalating tensions in the Middle East region."],
            ],
            inputs=input_text
        )

        with gr.Column(scale=1):
            output = gr.Markdown(label="Results")

    classify_btn.click(fn=predict_category, inputs=input_text, outputs=output)
    analyze_btn.click(fn=analyze_text_stats, inputs=input_text, outputs=output)

with gr.TabItem("📦 Batch Processing"):
    with gr.Row():
        with gr.Column():
            batch_input = gr.Textbox(
                label="Enter multiple articles (one per line)",
                placeholder="Article 1...\nArticle 2...\nArticle 3...",
                lines=12
            )
            batch_btn = gr.Button("📦 Classify All", variant="primary", size="lg")

        with gr.Column():
            batch_output = gr.Markdown(label="Batch Results")

    batch_btn.click(fn=predict_batch, inputs=batch_input, outputs=batch_output)

with gr.TabItem("⚖️ Compare Articles"):
    gr.Markdown("### Compare two articles side by side")

    with gr.Row():
        with gr.Column():
            article1 = gr.Textbox(label="Article 1", lines=6)
        with gr.Column():
            article2 = gr.Textbox(label="Article 2", lines=6)

    compare_btn = gr.Button("⚖️ Compare Articles", variant="primary")
    compare_output = gr.Markdown(label="Comparison Results")

    compare_btn.click(fn=compare_articles, inputs=[article1, article2], outputs=compare_output)

with gr.TabItem("📜 History"):
    gr.Markdown("### View recent predictions")

    with gr.Row():
        refresh_btn = gr.Button("🔄 Refresh History", variant="secondary")
        export_btn = gr.Button("📥 Export Results", variant="secondary")

    history_output = gr.Markdown(label="Prediction History")
    export_output = gr.Markdown(label="Export Status")

    refresh_btn.click(fn=get_prediction_history, outputs=history_output)
    export_btn.click(fn=export_results, outputs=export_output)
```

```
        with gr.TabItem("□ Performance Dashboard"):
            gr.Markdown(f"""
            ## Model Performance Metrics

            ### Test Set Results
            - *Accuracy:* {accuracy*100:.2f}%
            - *Precision:* {precision*100:.2f}%
            - *Recall:* {recall*100:.2f}%
            - *F1-Score:* {f1*100:.2f}%

            ### Model Architecture
            - *Algorithm:* Multinomial Naive Bayes
            - *Vectorization:* TF-IDF with bigrams
            - *Features:* 10,000 optimized features
            - *Training Samples:* {len(train_df):,}
            - *Test Samples:* {len(test_df):,}

            ### Dataset Statistics
            *Training Data:*
            - Total Articles: {len(train_df):,}
            - Average Text Length: {train_df['Text_Length'].mean():.0f} characters
            - Average Word Count: {train_df['Word_Count'].mean():.0f} words

            *Test Data:*
            - Total Articles: {len(test_df):,}
            - Average Text Length: {test_df['Text_Length'].mean():.0f} characters
            - Average Word Count: {test_df['Word_Count'].mean():.0f} words

            ### Category Breakdown
            - □ *World:* Politics, International Relations, Global Events
            - ⚽ *Sports:* Games, Matches, Athletes, Tournaments
            - □ *Business:* Economy, Stocks, Companies, Finance
            - □ *Science/Tech:* Science, Technology, Innovation, Research
            """)

        with gr.TabItem("ℹ️ About"):
            gr.Markdown("""
            ## About ClassiNews Pro

            ClassiNews Pro is an advanced machine learning system for automated news article categorization.

            ### Key Features
            - ✅ Real-time classification with confidence scores
            - 📦 Batch processing for multiple articles
            - Side-by-side article comparison
            - Detailed text analysis and statistics
            - Prediction history tracking
            - Export functionality for results
            - ✅ Visual confidence indicators

            ### Technology Stack
            - Machine Learning: Scikit-learn
            - NLP: NLTK, TF-IDF Vectorization
            - Visualization: Plotly, Matplotlib
            - Web Interface: Gradio
```

```
        - Dataset: AG News Classification Dataset

        ### Model Details
        The system uses Multinomial Naive Bayes classifier with:
        - TF-IDF vectorization (10,000 features)
        - Bigram analysis for better context
        - Stop words removal
        - Text preprocessing and normalization

        ### Performance Optimization
        - Bigram analysis for better context understanding
        - Optimized hyperparameters for maximum accuracy
        - Efficient text preprocessing pipeline
        """)

    gr.Markdown("---")
    gr.Markdown("*ClassiNews Pro* | Powered by Advanced ML | Built with Gradio")

print("Web interface created successfully with all features!")
```

 # STEP 10: Test with Sample Articles

```
print("\n" + "="*60)
print("🧪 TESTING WITH SAMPLE ARTICLES")
print("="*60)

test_articles = [
    "The government plans to increase funding for education and healthcare systems across the country.",
    "The football team won the championship after a thrilling match that went into overtime.",
    "Apple announces new iPhone with revolutionary features including foldable screen and AI assistant.",
    "Stock market reaches all-time high amid economic recovery and strong corporate earnings.",
    "United Nations discusses climate change policies at global summit with 150 countries participating.",
    "NBA finals game attracts millions of viewers worldwide breaking previous records.",
    "Microsoft acquires gaming company in multi-billion dollar deal to expand its entertainment division.",
    "New AI technology promises to revolutionize healthcare industry with faster diagnosis and treatment."
]

print("\nTesting sample articles:\n")
for i, article in enumerate(test_articles, 1):
    result = predict_category(article)
    predicted = result.split('*Predicted Category:*')[1].split('\n')[0].strip()
    print(f"📰 Article {i}:")
    print(f"   Text: {article[:80]}...")
    print(f"   Predicted: {predicted}")
    print()

# STEP 11: Save Model & Components

print("\n💾 Saving model and components...")

# Save model
joblib.dump(model, 'news_classifier_model.pkl')
```

```
# Save vectorizer
joblib.dump(vectorizer, 'tfidf_vectorizer.pkl')

# Save category mapping
with open('category_mapping.pkl', 'wb') as f:
    pickle.dump(category_map, f)

print("Model and components saved successfully!")
print("Files saved:")
print("  1. news_classifier_model.pkl")
print("  2. tfidf_vectorizer.pkl")
print("  3. category_mapping.pkl")

# STEP 12: Launch the Application

print("\n" + "="*60)
print(" LAUNCHING CLASSINEWS PRO APPLICATION")
print("="*60)

# Launch the interface
try:
    demo.launch(share=True)
    print(" Application launched successfully with sharing enabled!")
    print(" The web interface is now running.")
    print(" You can access it via the provided URL.")
except Exception as e:
    print(f"  Could not launch with share=True. Trying without sharing...")
    demo.launch()
    print(" Application launched locally!")

print("\n" + "="*60)
print(" ClassiNews Pro - Enhanced Version Completed!")
print("="*60)
print(" Features included:")
print("1. Real AG News dataset from Kaggle")
print("2. Separate train/test splits")
print("3. Interactive web interface with Gradio")
print("4.  Multiple visualizations (Word Clouds, Confusion Matrix, etc.)")
print("5. Feature importance analysis")
print("6.  Batch processing capability")
print("7.  Compare Articles feature")
print("8.  Prediction History tracking")
print("9.  Text Analysis & Statistics")
print("10.  Export Results to CSV")
print("11.  Performance metrics dashboard")
print("12.  Model persistence (saved files)")
print("="*60)
```