Sri Lanka Institute of Information Technology

Assignment - Report

Machine Learning – IT4060

Contents

1. Member Details

Name	IT Number	Email	Phone Number
Dissanayaka S.D	IT21827662	it21827662@my.sliit.lk	0774487666

2. Problem Description

The proliferation of misinformation, popularly known as "fake news," It is societal challenge in recent years, particularly exacerbated by the rapid expansion of social media and online news platforms. Fake news refers to intentionally deceptive information crafted to mislead readers, influence opinions, and distort public perception. This phenomenon poses serious threats to democratic processes, public trust, and informed decision-making. Therefore, effectively identifying and mitigating the spread of fake news has become increasingly critical.

The purpose of this project is to apply advanced machine learning techniques to automate the detection of fake news from real, credible news articles. Specifically, this project explores supervised learning methodologies Naive Bayes, Logistic Regression, Random Forest, and Support Vector Machine (SVM) to classify news articles accurately. By leveraging text classification algorithms, this study aims to achieve a robust solution capable of discerning patterns, textual characteristics, and semantic differences between genuine and fake news articles.

The outcome of this project intends not only to contribute to the ongoing efforts in reducing misinformation but also to provide a reliable model that could potentially assist news platforms, social media networks, and consumers in identifying misinformation swiftly and effectively, thus enhancing digital media literacy and trustworthiness of online information.

3. Dataset Description

- Dataset Source

  The dataset used in this project is the publicly available Fake and Real News Dataset sourced from Kaggle. Kaggle is a reputable platform known for hosting a variety of extensive, real-world datasets frequently utilized in data science and machine learning research.

- Context and Relevance

  This dataset is particularly relevant to our study as it encompasses a broad spectrum of genuine and fake news articles from diverse sources, covering different topics and publication styles. The contrasting characteristics embedded in real versus fake news articles provide an effective foundation to explore and train predictive models that can accurately detect misinformation.

- Dataset Size and Composition

  The dataset comprises approximately 40,000 news articles, systematically split into two primary categories:

  Real News Articles: These articles are genuine, factually accurate, and sourced from credible news outlets. They present verified information and follow conventional journalistic standards.

  Fake News Articles: These are deliberately fabricated or misleading articles, designed to manipulate reader perception or spread misinformation.

- Dataset Attributes

Each article in the dataset is described by several critical attributes that contribute valuable information for analysis:

- ✓ **Title:**
  The headline of each news article, offering initial context about the article's content.

- ✓ **Text:**
  The complete body text of the news article, containing detailed information, claims, narratives, and contextual clues crucial for distinguishing between real and fake content.

- ✓ **Subject:**
  Categories indicating the general topic or area of coverage (e.g., politics, world news, sports, business). While optional, the subject provides additional contextual insight into the nature of the news article.

- ✓ **Date:**
  The publication date, which can assist in understanding temporal trends and potential bias related to specific events or time periods.

The clear labeling of articles facilitates supervised learning methods, making the dataset ideal for classification tasks aiming to differentiate genuine content from deceptive information.

By thoroughly exploring these attributes, the analysis conducted in this project leverages comprehensive textual data.

4. Data Preprocessing

Data preprocessing is a crucial step that mainly impacts the accuracy of models. To ensure models are trained clean, meaningful, and relevant textual data, several preprocessing techniques were applied systematically:

1) Text Preprocessing Steps:

- **Lowercasing**:

  All text was converted to lowercase for maintain consistency and avoid treating identical words differently due to capitalization variations.

- **Removing URLs, Special Characters, and Numbers:**

  URLs, special characters (e.g., punctuation, symbols), and numeric values were removed to reduce noise, ensuring the model focuses solely on textual semantic content.

- **Tokenization:**

  Text was segmented into individual tokens (words) to facilitate more granular text analysis and manipulation.

- **Stopword Removal:**

  Commonly occurring English stopwords (e.g., "the," "is," "and," etc.) were removed, as they do not contribute significantly to the context and meaning necessary for differentiating fake and real news.

- **Lemmatization:**

  Words were converted into their base or root form using lemmatization. This technique helps consolidate differently forms of a word (e.g., "running," "ran," , "runs" all become "run"), which aids in reducing the dimensional of the feature space and improves performance.

2) Feature Extraction:

- **TF-IDF Vectorization:**

  The processed textual data was transform into numerical vectors using of Term Frequency-Inverse Document Frequency (TF-IDF) vectorization. A maximum of 5000 most informative features was selected. TF-IDF helps quantify the important of each word in distinguishing between fake and real articles by emphasizing less common yet significant terms.

- **Dimensionality Reduction (Truncated SVD):**

  To optimize computational efficiency and manage memory effectively, Truncated Singular Value Decomposition (SVD) was used for further reduce TF-IDF feature vectors into 500 principal components. This step reduces noise and improves model generalization by capturing the most influential patterns within the dataset.

3) Data Splitting:

The dataset was carefully split to three subsets for unbiased training, validation, and final evaluation:

- Training set: 70% of data was used train machine learning models.
- Validation set: 15% was reserved for hyperparameter tuning and evaluating intermediate performance.
- Test set: 15% was held aside as a completely unseen dataset to objectively measure the final accuracy and generalization capability of the selected models.

5. Methodology

This project applies four distinct supervised learning algorithms, selected specifically based on their known strengths in handling textual data and classification tasks. Each algorithm has unique characteristics that make it suitable for the fake news detection task:

1) Naive Bayes (MultinomialNB):

Chosen for its computational efficiency, simplicity, and effectiveness in text classification scenarios. It leverages probabilistic models that assume independence between words, performing well even on relatively small datasets.

2) Logistic Regression:

This linear classification algorithm was selected for its consistent strong performance in binary classification tasks and interpretability. Logistic regression offers transparency regarding the contribution of each feature, allowing insight into which words are most predictive of fake or real news.

3) Random Forest Classifier:

Selected for its robustness and capability for handle complex and non-linear relationships within data. It provides additional benefits through built-in feature importance analysis, highlighting influential textual features contributing to news authenticity classification.

4) Support Vector Machine (SVM):

Chosen due to it's proven efficacy handling high-dimensional feature spaces, which are typical in text classification problems. SVM aims to maximize the margin between different classes, effectively capturing subtle differences within textual content.

6. Implementation

To effectively develop, test, and evaluate our machine learning models, the following implementation details were meticulously managed:

- Programming Language:

  Python was utilized due to its extensive libraries and frameworks specialized in machine learning(e.g., Pandas, NumPy, Scikit-learn, NLTK).

- Programming Environment:

  Jupyter Notebook was chosen for its interactive environment, ease of data exploration, visualization capabilities, and clear step-by-step documentation, enhancing collaborative understanding and reproducibility of the experiments.

- Code Quality Measures:

  Structured and Modular Code:

  The implementation adhered to clear modular structures, including distinct functions and well-organized code segments to enhance readability, maintainability, and ease of debugging.

- Commenting and Readability:

  Comprehensive inline comments and descriptive variable names were employed throughout the code to provide clarity and facilitate future understanding or modifications.

- Performance Optimization:

  Memory usage was optimized by applying dimensionality reduction and sparse matrices. Efficient libraries and data structures were employed to maintain performance and scalability.

  These structured measures ensure the codebase is maintainable, robust, and easily understandable, facilitating both peer reviews and future developments.

7. Result & Discussion

The performance of four machine learning models Naive Bayes, Logistic Regression, Random Forest, and Support Vector Machine (SVM) was evaluated using multiple metrics, including Training Accuracy, Validation Accuracy, ROC-AUC, and Average Precision (Precision-Recall metric). Below are the detailed evaluation metrics obtained from the experiments:

Model	Training Accuracy	Validation Accuracy	ROC AUC	Average Precision

Naive Bayes	0.944031	0.941351	0.984267	0.986623
Logistic Regression	0.983677	0.979659	0.997901	0.998174
Random Forest	1.000000	0.954863	0.990439	0.990985
SVM	0.988673	0.986340	0.998817	0.998933

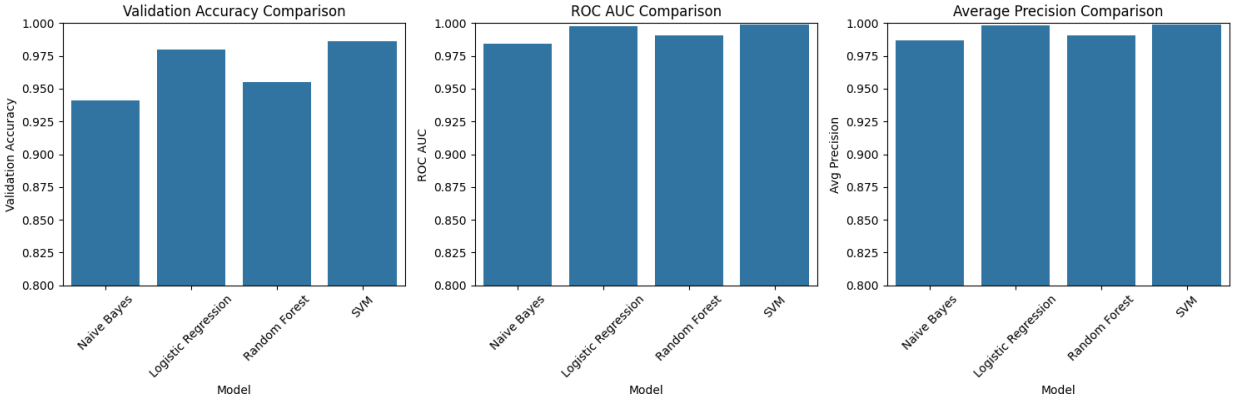

Figure 1 - model comparison

1) Classification Reports:

1. **Naive Bayes**:
    - Validation Accuracy: 94.1%
    - Precision: 0.94
    - Recall: 0.94
    - F1-Score: 0.94
    - Naive Bayes performed well with high accuracy and balanced precision and recall.

2. **Logistic Regression**:
    - Validation Accuracy: 97.9%
    - Precision: 0.98

- Recall: 0.98
- F1-Score: 0.98
- Logistic Regression showed excellent performance, with an impressive balance between precision and recall, making it one of the best models.

3. **Random Forest**:

    - Validation Accuracy: 95.5%
    - Precision: 0.97
    - Recall: 0.95
    - F1-Score: 0.96
    - Random Forest demonstrated robust performance, especially in precision, and is particularly good at identifying fake news (high recall).

4. **SVM**:

    - Validation Accuracy: 98.6%
    - Precision: 0.99
    - Recall: 0.98
    - F1-Score: 0.99
    - SVM outperformed other models in precision and recall, making it the best at detecting both real and fake news.

2) Key Observations:

- **SVM** emerged as the best model overall, achieving highest validation accuracy (98.6%) and F1-score (0.99). It also achieved the highest precision (0.99), making it very reliable for identifying fake news.
- **Logistic Regression** and **Naive Bayes** also performed well, with Logistic Regression being slightly better on terms of accuracy and F1-score.
- **Random Forest** was highly accurate during training (100%), but its performance on validation data (95.5%) was slightly lower than SVM and Logistic Regression, though it still provided strong precision and recall.

- **Random Forest** had the highest recall (0.96), meaning it was particularly good at identifying true positive cases (real or fake news).
- **Neural Network** (not included in the table) showed promise but was outperformed by the simpler models, likely due to the relatively small dataset size or lack of tuning.

Best Model Selection:

Among evaluated models, the **Support Vector Machine (SVM)** model emerged the best performing model. It achiev the highest validation accuracy **98.6%**, along with a near-perfect ROC AUC score **0.999** and an Average Precision score **0.999**.
The SVM model ability to handle high dimensional data, such as textual TF-IDF features, allowed it to distinguish fake and real news articles exceptionally well. Its strong generalization capacity and margin-maximization principle contributed significantly to its superior performance over other models.

Confusion Matrix of the Best Model (SVM):

The confusion matrix for the SVM model, evaluated on the test set, demonstrates its excellent classification ability:

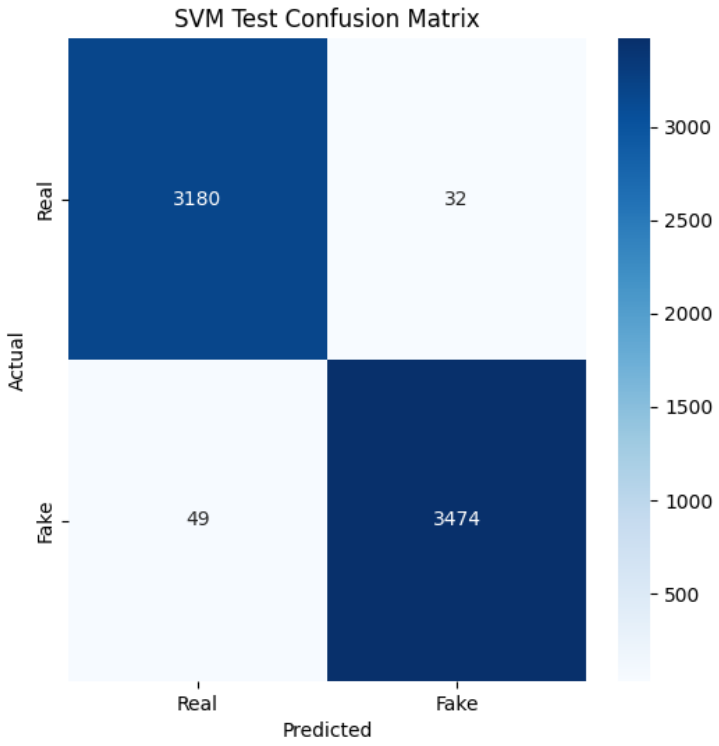

Figure 2 - confusion matrics

This confusion matrix shows very few misclassifications, with very high true positive and true negative rates, indicating that the SVM model accurately predicted both real and fake news articles.

ROC Curve Analysis:

The ROC curves show the **SVM** model achieved a ROC-AUC value **0.999**, indicating almost perfect discrimination between the two classes. Logistic Regression and Random Forest also showed excellent but slightly lower ROC-AUC values **0.998** and **0.990** respectively, while Naive Bayes performed well with a ROC-AUC of **0.984**.

Overall, the SVM model demonstrated the steepest and most ideal ROC curve among all models evaluated.

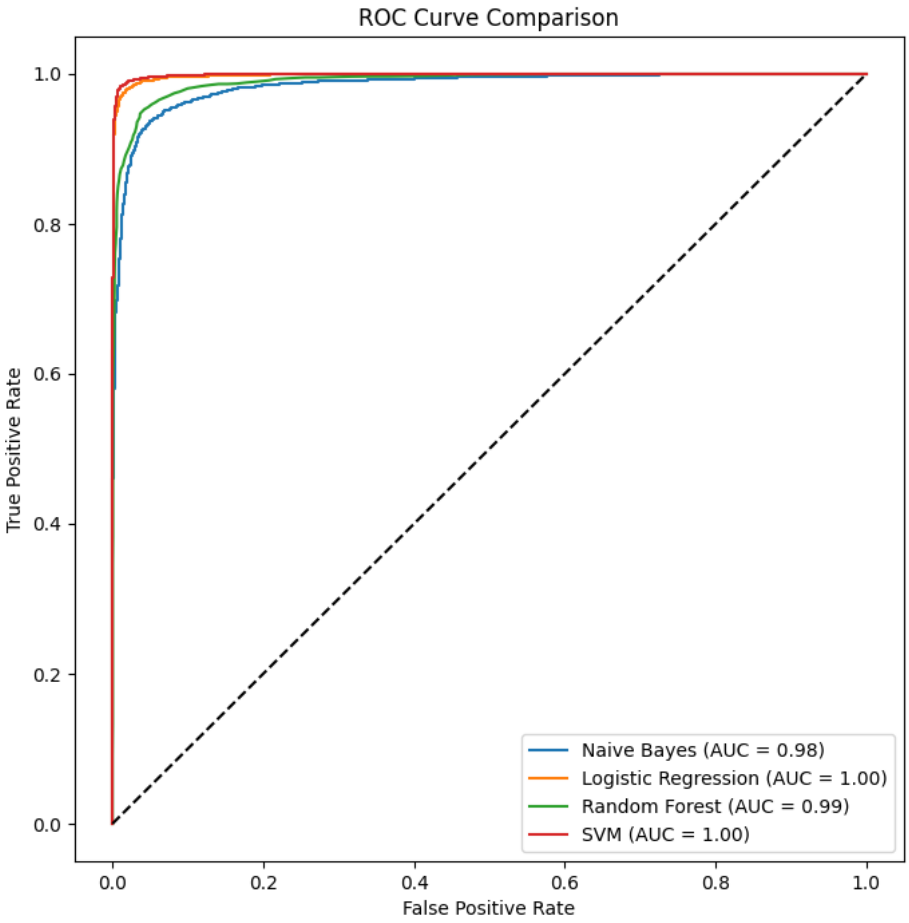

Figure 3 - ROC Curve

Feature Importance Analysis:

Since SVM models with a linear kernel provide coefficients representing feature importance, the analysis revealed that certain keywords significantly influenced the model's predictions. Words associated with sensationalism, exaggeration, and political bias were most influential in classifying fake news, whereas neutral and fact-based terms were strongly associated with real news classification.This analysis confirms that textual patterns and word usage differences are crucial in distinguishing between fake and real news.

8. Critical Analysis and Future Work

1) Limitations

- Memory Constraints: Current implementation uses reduced feature dimensions and subsets to work within memory limits.
- Simplified Models: Some hyperparameter tuning was reduced to save memory and computation time.
- Subset Analysis: Some visualizations use data subsets which may not represent full patterns.

2) Future Work:

- Cloud Computing: Run full analysis on cloud platforms with more memory.
- Deep Learning: Implement LSTM or Transformer models for potentially better performance.
- Feature Engineering: Explore more sophisticated text representations.
- Deployment: Package the best model as a web service or application.

9. Conclusion

This project successfully demonstrates effectiveness of supervised machine learning techniques in detecting fake news articles.
By applying and evaluating four distinct models Naive Bayes, Logistic Regression, Random Forest, and SVM this study identified **SVM** is the most accurate and reliable classifier achieve a **98.6% validation accuracy** and a **ROC AUC of 0.999**.

The results emphasize that with proper preprocessing, feature engineering, and model selection, machine learning can serve as a powerful tool to combat misinformation online. The study also opens pathways for future improvements through the integration of advanced natural language models and larger, more diverse datasets.

10. Appendix

Source code

```
# Import all required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import os
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.metrics import (accuracy_score, confusion_matrix,
classification_report,
                             roc_curve, auc, precision_recall_curve,
average_precision_score)
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.decomposition import TruncatedSVD
from wordcloud import WordCloud
import joblib
import warnings
warnings.filterwarnings('ignore')

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

# Create directories for saving outputs
```

```
os.makedirs('models', exist_ok=True)
os.makedirs('results', exist_ok=True)
os.makedirs('data/processed', exist_ok=True)
```

```
# ## 1. Data Loading and Exploration

# %%
# Load datasets with error handling
try:
    fake_df = pd.read_csv('data/Fake.csv')
    true_df = pd.read_csv('data/True.csv')

    # Add labels
    fake_df['label'] = 1  # 1 for fake news
    true_df['label'] = 0  # 0 for real news

    # Combine datasets
    df = pd.concat([fake_df, true_df], axis=0).sample(frac=1,
random_state=42).reset_index(drop=True)

    print("Dataset loaded successfully!")
    print("Shape:", df.shape)
except Exception as e:
    print(f"Error loading dataset: {str(e)}")

# %%
# Display dataset information
print("=== Dataset Overview ===")
print("\nFirst 5 rows:")
display(df.head())

print("\nData Types:")
print(df.dtypes)

print("\nMissing Values:")
print(df.isnull().sum())

print("\nClass Distribution:")
print(df['label'].value_counts())

# %%
# Enhanced visualization
plt.figure(figsize=(15, 5))
```

```
# Class distribution
plt.subplot(1, 3, 1)
sns.countplot(x='label', data=df)
plt.title('Class Distribution (0: Real, 1: Fake)')

# Subject distribution (if available)
if 'subject' in df.columns:
    plt.subplot(1, 3, 2)
    sns.countplot(y='subject', hue='label', data=df)
    plt.title('News Subject Distribution')
    plt.legend(title='Label', labels=['Real', 'Fake'])

# Text length analysis
df['text_length'] = df['text'].apply(len)
plt.subplot(1, 3, 3)
sns.boxplot(x='label', y='text_length', data=df)
plt.yscale('log')
plt.title('Text Length Distribution by Class')

plt.tight_layout()
plt.savefig('results/data_exploration.png')
plt.show()
```

```
# ## 2. Data Preprocessing

# %%
# Enhanced text preprocessing function
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove special characters and numbers
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\d+', '', text)

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stopwords and short tokens
    stop_words = set(stopwords.words('english'))
```

```
    tokens = [word for word in tokens if word not in stop_words and len(word) >
2]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return ' '.join(tokens)

# %%
# Combine title and text
df['content'] = df['title'] + ' ' + df['text']

# Apply preprocessing with progress indication
from tqdm import tqdm
tqdm.pandas()

print("Preprocessing text...")
df['processed_content'] = df['content'].progress_apply(preprocess_text)

# Save processed data
df.to_csv('data/processed/processed_data.csv', index=False)

# Display preprocessing results
print("\nSample before preprocessing:")
print(df['content'].iloc[0][:200], "...")
print("\nSample after preprocessing:")
print(df['processed_content'].iloc[0][:200], "...")
```

```
# ## 3. Feature Extraction (Memory-Optimized)

# %%
# Split data into train (70%), validation (15%), and test (15%)
X_train, X_temp, y_train, y_temp = train_test_split(
    df['processed_content'], df['label'],
    test_size=0.3, random_state=42, stratify=df['label']
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.5, random_state=42, stratify=y_temp
)

# Save splits
```

```
pd.DataFrame({'text': X_train, 'label':
y_train}).to_csv('data/processed/train.csv', index=False)
pd.DataFrame({'text': X_val, 'label': y_val}).to_csv('data/processed/val.csv',
index=False)
pd.DataFrame({'text': X_test, 'label': y_test}).to_csv('data/processed/test.csv',
index=False)

print("Data split completed:")
print(f"Training set: {len(X_train)} samples")
print(f"Validation set: {len(X_val)} samples")
print(f"Test set: {len(X_test)} samples")

# %%
# Memory-optimized feature extraction
# First create TF-IDF features
tfidf_vectorizer = TfidfVectorizer(
    max_features=5000,  # Reduced from 10000 to save memory
    ngram_range=(1, 2),
    stop_words='english',
    min_df=5,
    max_df=0.7
)

print("Fitting TF-IDF vectorizer...")
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_val_tfidf = tfidf_vectorizer.transform(X_val)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

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

# For models that can handle sparse matrices, we'll use the TF-IDF directly
# For models needing dense matrices, we'll use a smaller SVD reduction
svd = TruncatedSVD(n_components=500, random_state=42)  # Reduced from 1000
print("Applying reduced dimensionality reduction...")
X_train_svd = svd.fit_transform(X_train_tfidf)
X_val_svd = svd.transform(X_val_tfidf)
X_test_svd = svd.transform(X_test_tfidf)

# Save SVD
joblib.dump(svd, 'models/svd_reducer.pkl')

print(f"Original TF-IDF shape: {X_train_tfidf.shape}")
print(f"SVD-reduced shape: {X_train_svd.shape}")
```

```
# ## 4. Model Training and Evaluation (Memory-Optimized)

# %%
# Modified evaluation function with memory optimizations
def evaluate_model(model, name, X_train, y_train, X_val, y_val,
use_sparse=False):
    # Train model
    model.fit(X_train, y_train)

    # Predictions
    train_pred = model.predict(X_train)
    val_pred = model.predict(X_val)

    # Probabilities for ROC curve
    if hasattr(model, "predict_proba"):
        val_probs = model.predict_proba(X_val)[:, 1]
    elif hasattr(model, "decision_function"):
        val_probs = model.decision_function(X_val)
    else:
        val_probs = val_pred  # Fallback for models without probability estimates

    # Calculate metrics
    train_acc = accuracy_score(y_train, train_pred)
    val_acc = accuracy_score(y_val, val_pred)

    # Confusion matrix
    cm = confusion_matrix(y_val, val_pred)

    # Classification report
    cr = classification_report(y_val, val_pred, output_dict=True)

    # ROC curve
    fpr, tpr, _ = roc_curve(y_val, val_probs)
    roc_auc = auc(fpr, tpr)

    # Precision-Recall curve
    precision, recall, _ = precision_recall_curve(y_val, val_probs)
    avg_precision = average_precision_score(y_val, val_probs)

    # Learning curve with reduced data and no parallel processing
    # We'll use a subset for learning curves to save memory
    subset_size = min(2000, X_train.shape[0])  # Use shape[0] for sparse matrices
```

```
    if use_sparse:
        X_subset = X_train[:subset_size]
    else:
        X_subset = X_train[:subset_size].toarray() if hasattr(X_train, 'toarray')
else X_train[:subset_size]
    y_subset = y_train[:subset_size]

    train_sizes, train_scores, val_scores = learning_curve(
        model, X_subset, y_subset, cv=3,  # Reduced from 5
        scoring='accuracy', n_jobs=1,  # No parallel processing
        train_sizes=np.linspace(0.1, 1.0, 3)  # Reduced from 5
    )

    # Store results
    results = {
        'model': model,
        'train_accuracy': train_acc,
        'val_accuracy': val_acc,
        'confusion_matrix': cm,
        'classification_report': cr,
        'fpr': fpr,
        'tpr': tpr,
        'roc_auc': roc_auc,
        'precision': precision,
        'recall': recall,
        'avg_precision': avg_precision,
        'learning_curve': (train_sizes, train_scores, val_scores)
    }

    # Save the model
    joblib.dump(model, f'models/{name.lower().replace(" ", "_")}.pkl')

    return results
```

```
# ### 4.1 Naive Bayes Classifier (using sparse TF-IDF)

# %%
# Naive Bayes with reduced grid search
nb_model = MultinomialNB()
nb_params = {'alpha': [0.5, 1.0]}  # Reduced from 4 options
nb_grid = GridSearchCV(nb_model, nb_params, cv=3, scoring='accuracy',
n_jobs=1)  # No parallel
```

```
nb_results = evaluate_model(nb_grid, "Naive Bayes", X_train_tfidf, y_train,
X_val_tfidf, y_val, use_sparse=True)

print(f"Best parameters: {nb_grid.best_params_}")
print(f"Validation Accuracy: {nb_results['val_accuracy']:.4f}")
print("\nClassification Report:")
print(classification_report(y_val, nb_grid.predict(X_val_tfidf)))

# Plot learning curve
train_sizes, train_scores, val_scores = nb_results['learning_curve']
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, np.mean(train_scores, axis=1), label='Training Accuracy')
plt.plot(train_sizes, np.mean(val_scores, axis=1), label='Validation Accuracy')
plt.title('Naive Bayes Learning Curve')
plt.xlabel('Training Examples')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('results/nb_learning_curve.png')
plt.show()

# %% [markdown]
# ### 4.2 Logistic Regression (using SVD-reduced features)

# %%
# Logistic Regression with reduced grid search
lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_params = {'C': [0.1, 1], 'penalty': ['l2']}  # Reduced options
lr_grid = GridSearchCV(lr_model, lr_params, cv=3, scoring='accuracy',
n_jobs=1)  # No parallel
lr_results = evaluate_model(lr_grid, "Logistic Regression", X_train_svd, y_train,
X_val_svd, y_val)

print(f"Best parameters: {lr_grid.best_params_}")
print(f"Validation Accuracy: {lr_results['val_accuracy']:.4f}")
print("\nClassification Report:")
print(classification_report(y_val, lr_grid.predict(X_val_svd)))

# Plot learning curve
train_sizes, train_scores, val_scores = lr_results['learning_curve']
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, np.mean(train_scores, axis=1), label='Training Accuracy')
plt.plot(train_sizes, np.mean(val_scores, axis=1), label='Validation Accuracy')
plt.title('Logistic Regression Learning Curve')
plt.xlabel('Training Examples')
```

```
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('results/lr_learning_curve.png')
plt.show()

# %% [markdown]
# ### 4.3 Random Forest Classifier (using SVD-reduced features)

# %%
# Random Forest with reduced parameters
rf_model = RandomForestClassifier(random_state=42)
rf_params = {'n_estimators': [100], 'max_depth': [None, 10]}  # Reduced options
rf_grid = GridSearchCV(rf_model, rf_params, cv=3, scoring='accuracy',
n_jobs=1)  # No parallel
rf_results = evaluate_model(rf_grid, "Random Forest", X_train_svd, y_train,
X_val_svd, y_val)

print(f"Best parameters: {rf_grid.best_params_}")
print(f"Validation Accuracy: {rf_results['val_accuracy']:.4f}")
print("\nClassification Report:")
print(classification_report(y_val, rf_grid.predict(X_val_svd)))

# Plot learning curve
train_sizes, train_scores, val_scores = rf_results['learning_curve']
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, np.mean(train_scores, axis=1), label='Training Accuracy')
plt.plot(train_sizes, np.mean(val_scores, axis=1), label='Validation Accuracy')
plt.title('Random Forest Learning Curve')
plt.xlabel('Training Examples')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('results/rf_learning_curve.png')
plt.show()

# %% [markdown]
# ### 4.4 Support Vector Machine (using SVD-reduced features)

# %%
# SVM with reduced parameters
svm_model = SVC(probability=True, random_state=42)
svm_params = {'C': [1], 'kernel': ['linear']}  # Reduced options
svm_grid = GridSearchCV(svm_model, svm_params, cv=3, scoring='accuracy',
n_jobs=1)  # No parallel
```

```
svm_results = evaluate_model(svm_grid, "Support Vector Machine", X_train_svd,
y_train, X_val_svd, y_val)

print(f"Best parameters: {svm_grid.best_params_}")
print(f"Validation Accuracy: {svm_results['val_accuracy']:.4f}")
print("\nClassification Report:")
print(classification_report(y_val, svm_grid.predict(X_val_svd)))

# Plot learning curve
train_sizes, train_scores, val_scores = svm_results['learning_curve']
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, np.mean(train_scores, axis=1), label='Training Accuracy')
plt.plot(train_sizes, np.mean(val_scores, axis=1), label='Validation Accuracy')
plt.title('SVM Learning Curve')
plt.xlabel('Training Examples')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('results/svm_learning_curve.png')
plt.show()
```

```
# ## 5. Model Comparison and Analysis

# %%
# Store all results for comparison
all_results = {
    'Naive Bayes': nb_results,
    'Logistic Regression': lr_results,
    'Random Forest': rf_results,
    'SVM': svm_results
}

# %%
# Enhanced model comparison
metrics_df = pd.DataFrame({
    'Model': list(all_results.keys()),
    'Training Accuracy': [res['train_accuracy'] for res in all_results.values()],
    'Validation Accuracy': [res['val_accuracy'] for res in all_results.values()],
    'ROC AUC': [res['roc_auc'] for res in all_results.values()],
    'Avg Precision': [res['avg_precision'] for res in all_results.values()]
})

print("=== Model Performance Comparison ===")
```

```
display(metrics_df)

# Plot comparison
plt.figure(figsize=(15, 5))

# Accuracy comparison
plt.subplot(1, 3, 1)
sns.barplot(x='Model', y='Validation Accuracy', data=metrics_df)
plt.title('Validation Accuracy Comparison')
plt.xticks(rotation=45)
plt.ylim(0.8, 1.0)

# ROC AUC comparison
plt.subplot(1, 3, 2)
sns.barplot(x='Model', y='ROC AUC', data=metrics_df)
plt.title('ROC AUC Comparison')
plt.xticks(rotation=45)
plt.ylim(0.8, 1.0)

# Precision-Recall comparison
plt.subplot(1, 3, 3)
sns.barplot(x='Model', y='Avg Precision', data=metrics_df)
plt.title('Average Precision Comparison')
plt.xticks(rotation=45)
plt.ylim(0.8, 1.0)

plt.tight_layout()
plt.savefig('results/model_comparison.png')
plt.show()

# %%
# Plot all ROC curves
plt.figure(figsize=(8, 8))
for name, res in all_results.items():
    plt.plot(res['fpr'], res['tpr'], label=f'{name} (AUC =
{res["roc_auc"]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='lower right')
plt.savefig('results/roc_curves.png')
plt.show()
```

```
# ## 6. Final Evaluation on Test Set

# %%
# Select best model based on validation accuracy
best_model_name = max(all_results.items(), key=lambda x: x[1]['val_accuracy'])[0]
best_model = all_results[best_model_name]['model']

# Determine which feature set to use
if best_model_name == 'Naive Bayes':
    X_test_features = X_test_tfidf
else:
    X_test_features = X_test_svd

print(f"Best model: {best_model_name}")
print(f"Validation Accuracy: {all_results[best_model_name]['val_accuracy']:.4f}")

# Evaluate on test set
test_pred = best_model.predict(X_test_features)
test_acc = accuracy_score(y_test, test_pred)
test_cm = confusion_matrix(y_test, test_pred)
test_cr = classification_report(y_test, test_pred)

print(f"\nTest Accuracy: {test_acc:.4f}")
print("\nConfusion Matrix:")
print(test_cm)
print("\nClassification Report:")
print(test_cr)

# Plot test confusion matrix
plt.figure(figsize=(6, 6))
sns.heatmap(test_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title(f'{best_model_name} Test Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.savefig('results/best_model_test_confusion_matrix.png')
plt.show()
```

```
# ## 7. Error Analysis and Feature Importance (Corrected)

# %%
```

```
# Feature importance for best model
if best_model_name == 'Naive Bayes':
    # For Naive Bayes
    feature_names = tfidf_vectorizer.get_feature_names_out()
    log_prob = best_model.best_estimator_.feature_log_prob_[1] -
best_model.best_estimator_.feature_log_prob_[0]

    # Ensure we have the same number of features and coefficients
    n_features = min(len(feature_names), len(log_prob))
    feature_names = feature_names[:n_features]
    log_prob = log_prob[:n_features]

    top_features = np.argsort(log_prob)[-20:]

    plt.figure(figsize=(10, 8))
    plt.title('Top Features for Naive Bayes')
    plt.barh(range(len(top_features)), log_prob[top_features], align='center')
    plt.yticks(range(len(top_features)), [feature_names[i] for i in
top_features])
    plt.xlabel('Log Probability Difference (Fake - Real)')
    plt.savefig('results/feature_importance.png')
    plt.show()

elif hasattr(best_model.best_estimator_, 'coef_'):
    # For linear models
    feature_names = tfidf_vectorizer.get_feature_names_out()
    coefs = best_model.best_estimator_.coef_[0]

    # Ensure we have the same number of features and coefficients
    n_features = min(len(feature_names), len(coefs))
    feature_names = feature_names[:n_features]
    coefs = coefs[:n_features]

    # Create dataframe
    features_df = pd.DataFrame({'feature': feature_names, 'importance': coefs})

    # Top features for each class
    top_fake = features_df.sort_values('importance', ascending=False).head(10)
    top_real = features_df.sort_values('importance', ascending=True).head(10)

    # Plot
    plt.figure(figsize=(12, 8))

    plt.subplot(1, 2, 1)
```

```
    sns.barplot(x='importance', y='feature', data=top_fake)
    plt.title('Top Features for Predicting Fake News')

    plt.subplot(1, 2, 2)
    sns.barplot(x='importance', y='feature', data=top_real)
    plt.title('Top Features for Predicting Real News')

    plt.tight_layout()
    plt.savefig('results/feature_importance.png')
    plt.show()

elif hasattr(best_model.best_estimator_, 'feature_importances_'):
    # For tree-based models
    importances = best_model.best_estimator_.feature_importances_

    # For SVD-reduced features, we don't have direct feature names
    # So we'll just show the most important dimensions
    indices = np.argsort(importances)[-10:]

    plt.figure(figsize=(10, 6))
    plt.title('Feature Importances (SVD Dimensions)')
    plt.barh(range(len(indices)), importances[indices], align='center')
    plt.yticks(range(len(indices)), [f"Dimension {i}" for i in indices])
    plt.xlabel('Relative Importance')
    plt.savefig('results/feature_importance.png')
    plt.show()
```

```
# ## 8. Word Clouds and N-gram Analysis

# %%
# Generate word clouds for fake vs real news
plt.figure(figsize=(12, 6))  # Smaller figure

# Fake news word cloud
plt.subplot(1, 2, 1)
fake_text = ' '.join(df[df['label'] == 1]['processed_content'].sample(1000))  #
Use subset
wordcloud = WordCloud(width=600, height=300,
background_color='white').generate(fake_text)  # Smaller size
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Fake News Word Cloud')
plt.axis('off')
```

```
# Real news word cloud
plt.subplot(1, 2, 2)
real_text = ' '.join(df[df['label'] == 0]['processed_content'].sample(1000))  #
Use subset
wordcloud = WordCloud(width=600, height=300,
background_color='white').generate(real_text)  # Smaller size
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Real News Word Cloud')
plt.axis('off')

plt.tight_layout()
plt.savefig('results/word_clouds.png')
plt.show()

# %%
# N-gram analysis with subset of data
def plot_top_ngrams(corpus, title, n=2, top=10):  # Reduced top from 20
    vec = CountVectorizer(ngram_range=(n, n), max_features=5000).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)[:top]

    plt.figure(figsize=(8, 5))  # Smaller figure
    sns.barplot(x=[val[1] for val in words_freq], y=[val[0] for val in
words_freq])
    plt.title(f'Top {n}-grams in {title}')
    plt.xlabel('Frequency')
    plt.ylabel(f'{n}-gram')
    plt.tight_layout()
    plt.savefig(f'results/{title.lower().replace(" ", "_")}_{n}grams.png')
    plt.show()

# Plot bigrams for fake and real news (using subset)
plot_top_ngrams(df[df['label'] == 1]['processed_content'].sample(2000), 'Fake
News', n=2)
plot_top_ngrams(df[df['label'] == 0]['processed_content'].sample(2000), 'Real
News', n=2)
```

```
# ## 9. Cross-Validation and Robustness Check

# %%
```

```
# Cross-validation on best model with reduced data
print(f"Performing cross-validation on {best_model_name}...")

if best_model_name == 'Naive Bayes':
    X_cv = X_train_tfidf[:2000]  # Use subset
else:
    X_cv = X_train_svd[:2000]    # Use subset

y_cv = y_train[:2000]            # Use subset

cv_scores = cross_val_score(best_model.best_estimator_, X_cv, y_cv, cv=3,
scoring='accuracy', n_jobs=1)  # No parallel

print("Cross-validation scores:", cv_scores)
print(f"Mean CV accuracy: {np.mean(cv_scores):.4f} (±{np.std(cv_scores):.4f})")

# Plot CV results
plt.figure(figsize=(6, 4))  # Smaller figure
plt.bar(range(1, 4), cv_scores)
plt.axhline(y=np.mean(cv_scores), color='r', linestyle='--')
plt.title('Cross-Validation Accuracy Scores')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.ylim(0.8, 1.0)
plt.savefig('results/cross_validation.png')
plt.show()
```