Final project: Fake news detection with unsupervised algorithms

Fake news detection has become one of the most pressing challenges in today's digital age, significantly impacting critical areas such as politics, health, and education. The rapid spread of misinformation can lead to widespread confusion, influencing public opinion and decision-making in harmful ways. Given the importance of addressing this issue, finding effective methods to detect and prevent the dissemination of fake news is crucial.

This project aims to explore various machine learning techniques, particularly focusing on the potential of unsupervised models, to identify and classify fake news articles. By analyzing a labeled dataset originally intended for deep learning models, we investigate whether unsupervised approaches can offer viable solutions for detecting misinformation. Through this exploration, the project seeks to contribute to the broader effort of combating fake news and promoting accurate information in society. Since labeled data is often scarce, it is especially important to evaluate how effective a model can be when no labeled data is used.

Objectives

The objective of this project is to develop and evaluate various machine learning models for the detection and classification of fake news articles. The project aims to compare the performance of both supervised and unsupervised learning techniques, including Non-Negative Matrix Factorization (NMF), KMeans Clustering, Hierarchical Clustering, and Support Vector Machines (SVM), by analyzing their accuracy and root mean square error (RMSE). Additionally, the project explores the impact of dimensionality reduction techniques such as PCA and the effectiveness of hyperparameter optimization in enhancing model performance. The ultimate goal is to identify the most reliable model for distinguishing between true and fake news articles based on textual data.

Data description

The data for this project was obtained from Kaggle. Although it was originally intended for training deep learning models, with articles labeled as "fake" and "true," we will use it to explore the effectiveness of unsupervised models.

Source: https://www.kaggle.com/code/therealsampat/fake-news-detection/input

The data was collected and labeled in the context of the 2016 U.S. election.

Import Libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA, NMF
from sklearn.manifold import TSNE, MDS
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.preprocessing import StandardScaler, Binarizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, mean_squared_error
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.svm import SVC
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
import string
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

import nltk

# Download the necessary NLTK data files
#nltk.download('punkt')
#nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

import warnings
warnings.filterwarnings('ignore')
```

Data loading

This section of the code loads the fake and true news datasets, adds labels to differentiate between them, and then combines the datasets into a single DataFrame. The dataset is optionally balanced by downsampling the larger class, and a smaller sample is created for use in model building and analysis.

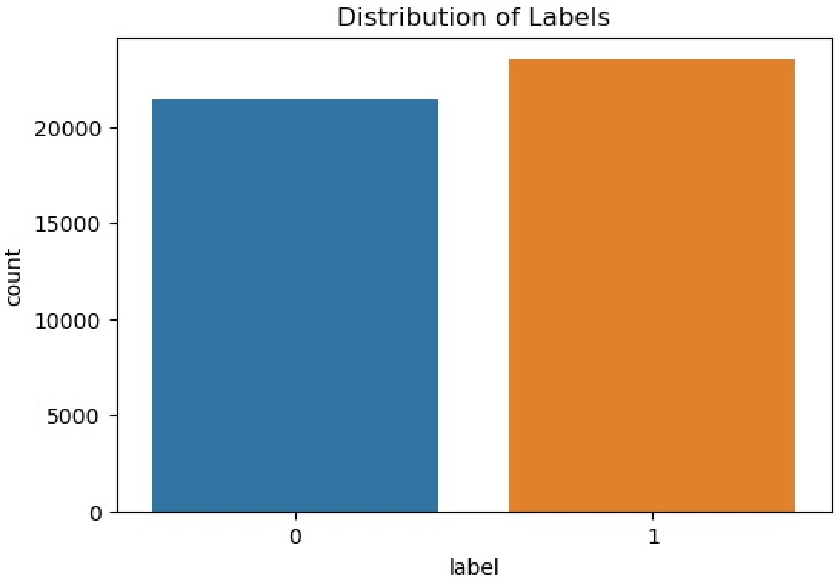
In [2]:
```
# Load the datasets
fake_df = pd.read_csv(r'C:\Users\pzuloaga\Desktop\Uns Algor\Final\Data\Fake.csv')
```

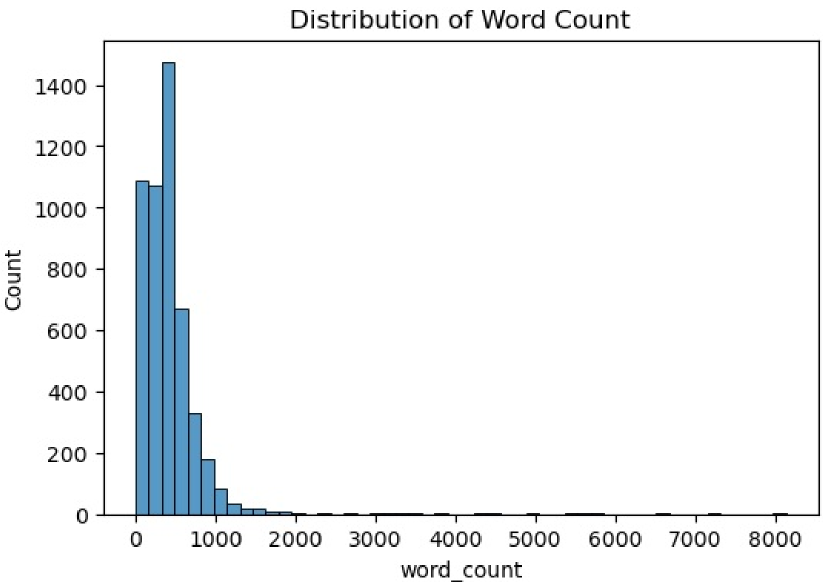
```python
true_df = pd.read_csv(r'C:\Users\pzuloaga\Desktop\Uns Algor\Final\Data\True.csv')

# Add labels
fake_df['label'] = 1
true_df['label'] = 0

# Combine the datasets
df = pd.concat([fake_df, true_df], ignore_index=True)

# Optional: Balance the dataset by downsampling the larger class
min_len = min(len(fake_df), len(true_df))
fake_df_downsampled = fake_df.sample(min_len)
true_df_downsampled = true_df.sample(min_len)

df_balanced = pd.concat([fake_df_downsampled, true_df_downsampled], ignore_index=True)

# Optional: Create a smaller sample for model building
sample_size = 5000  # Set this variable to change the sample size
df_sample = df_balanced.sample(sample_size, random_state=42)
```

Exploratory Data Analysis

In [3]:
```python
# Basic EDA
print("Dataframe shape:", df.shape)
print("Dataframe description:\n", df.describe())

# Display the first two rows of the dataframe as a sample
print("Sample data (first two rows):")
print(df.head(2))

# Visualizations
plt.figure(figsize=(6, 4))
sns.countplot(x='label', data=df)
plt.title('Distribution of Labels')
plt.show()

# Word count distribution
df_sample['word_count'] = df_sample['text'].apply(lambda x: len(x.split()))
plt.figure(figsize=(6, 4))
sns.histplot(df_sample['word_count'], bins=50)
plt.title('Distribution of Word Count')
plt.show()
```

```
Dataframe shape: (44898, 5)
Dataframe description:
               label
count  44898.000000
mean       0.522985
std        0.499477
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        1.000000
Sample data (first two rows):
                                               title  \
0   Donald Trump Sends Out Embarrassing New Year’...
1   Drunk Bragging Trump Staffer Started Russian ...

                                                text subject  \
0  Donald Trump just couldn t wish all Americans ...    News
1  House Intelligence Committee Chairman Devin Nu...    News

                date  label
0  December 31, 2017      1
1  December 31, 2017      1
```





The EDA reveals that the dataset consists of 44,898 entries with 5 columns. The label distribution shows a nearly equal split between classes, as indicated by a mean close to 0.5. The histogram suggests that the data is not severely unbalanced. The first two rows of the sample data confirm that the dataset contains both the title and text of articles, along with their respective subjects, dates, and labels.

Data preprocessing and cleaning

This section focuses on text preprocessing, which is a crucial step in preparing textual data for analysis. The `clean_text` function is defined to standardize the text by performing the following steps:

1. Convert the text to lowercase to ensure uniformity.
2. Remove punctuation and numbers, as they often do not contribute to the meaning of the text in this context.
3. Tokenize the text into individual words.

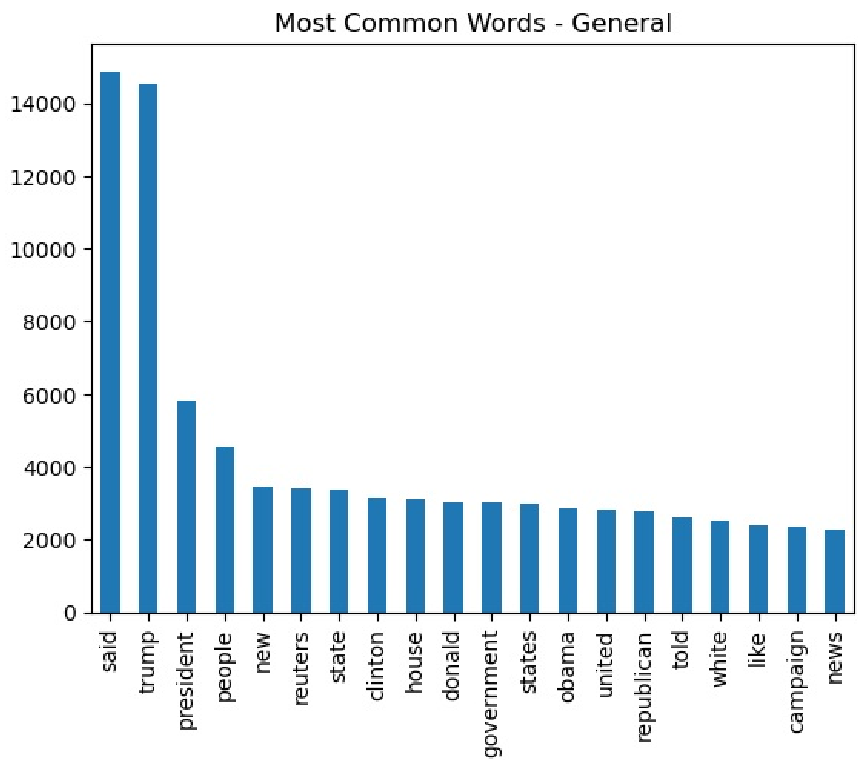
4. Filter out non-alphabetical characters and stopwords (common words that typically do not carry significant meaning, such as "the" and "and").

The cleaned text is then stored in a new column called `clean_text`.

After preprocessing, the `CountVectorizer` is applied to the cleaned text to identify and count the 20 most common words in the dataset, excluding English stopwords. The frequencies of these words are visualized in a bar plot, providing insights into the most prevalent terms across the sample dataset. -->

In [4]:
```
# Text Preprocessing
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()
    text = re.sub(f'[{string.punctuation}]', '', text)  # Remove punctuation
    text = re.sub(r'\d+', '', text)  # Remove numbers
    words = word_tokenize(text)
    words = [word for word in words if word.isalpha()]  # Remove non-alphabetical characters
    words = [word for word in words if word not in stop_words]  # Remove stopwords
    return ' '.join(words)

df_sample['clean_text'] = df_sample['text'].apply(clean_text)
```

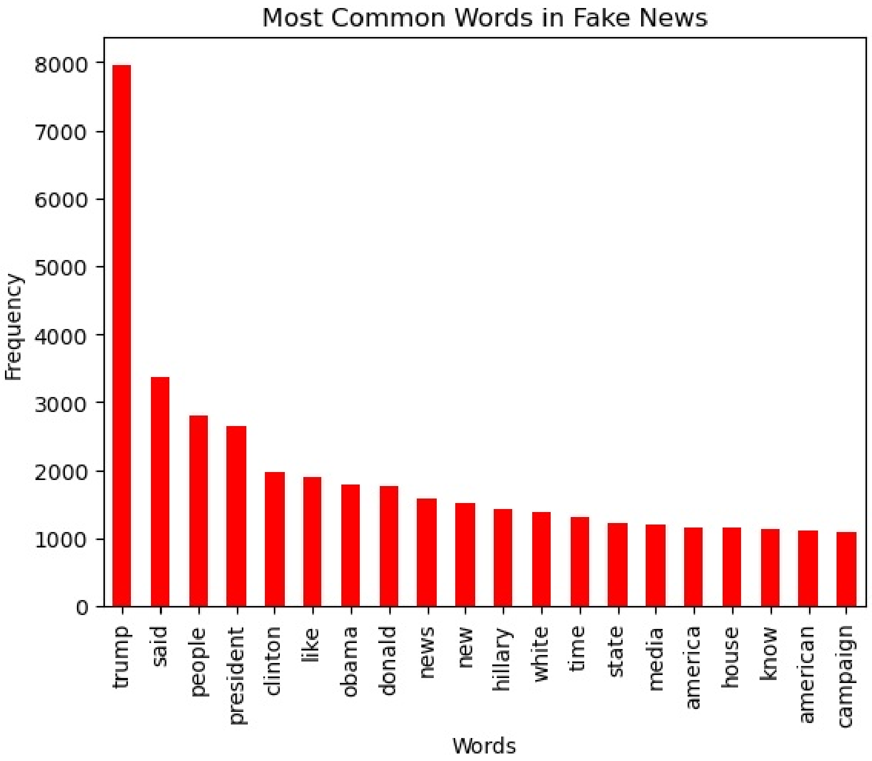
In [5]:
```
# Display most common words
vectorizer = CountVectorizer(stop_words='english', max_features=20)
X_counts = vectorizer.fit_transform(df_sample['clean_text'])
common_words = pd.DataFrame(X_counts.toarray(), columns=vectorizer.get_feature_names_out()).sum().sort_values(a
common_words.plot(kind='bar')
plt.title('Most Common Words - General')
plt.show()
```

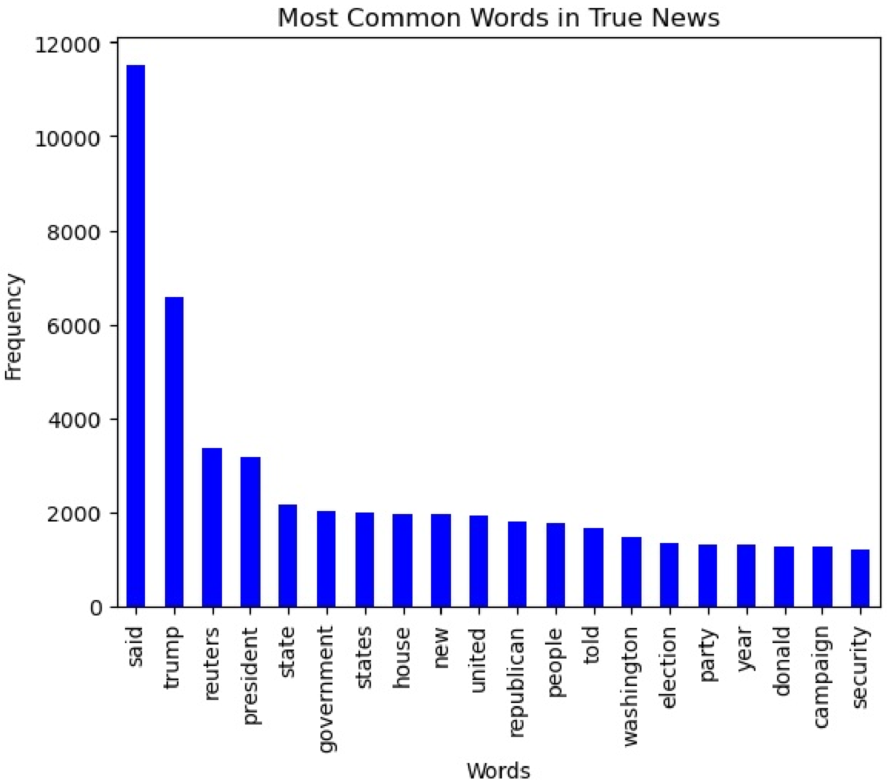


We will now explore how different are the most common words for each of the labels to observe if there is a clear differentiation

In [26]:
```
# Separate the data by label
df_fake = df_sample[df_sample['label'] == 1]
df_true = df_sample[df_sample['label'] == 0]

# Display most common words for Fake News
vectorizer_fake = CountVectorizer(stop_words='english', max_features=20)
X_counts_fake = vectorizer_fake.fit_transform(df_fake['clean_text'])
common_words_fake = pd.DataFrame(X_counts_fake.toarray(), columns=vectorizer_fake.get_feature_names_out()).sum(

#plt.figure(figsize=(10, 6))
common_words_fake.plot(kind='bar', color='red')
plt.title('Most Common Words in Fake News')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.show()

# Display most common words for True News
vectorizer_true = CountVectorizer(stop_words='english', max_features=20)
X_counts_true = vectorizer_true.fit_transform(df_true['clean_text'])
common_words_true = pd.DataFrame(X_counts_true.toarray(), columns=vectorizer_true.get_feature_names_out()).sum(
```

```
#plt.figure(figsize=(10, 6))
common_words_true.plot(kind='bar', color='blue')
plt.title('Most Common Words in True News')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.show()
```





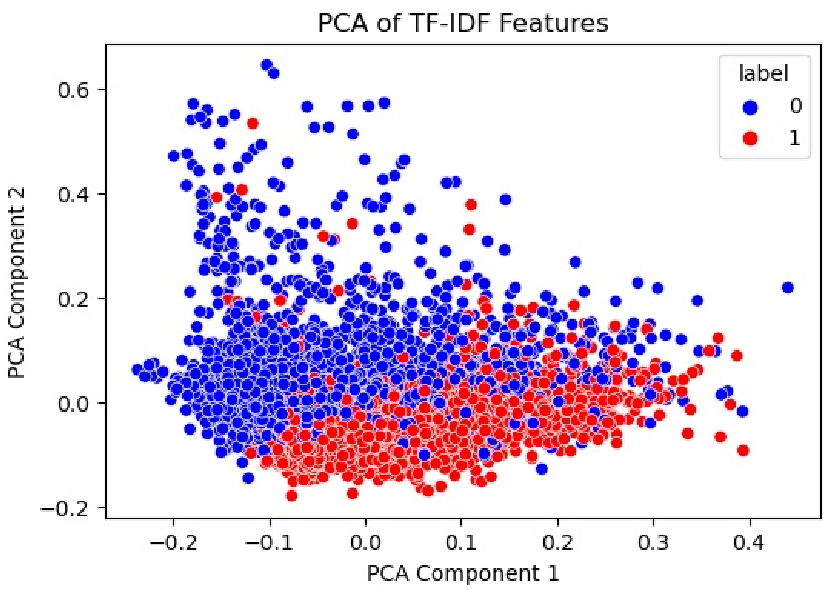
From the two histograms above, we can already realized the are several words that are repeated in the two datasets and this may difficult the analysis when all the text are processed together. Since there is no clear differentiation at this level. So we will explore if we can recognize some pattern after PCA.

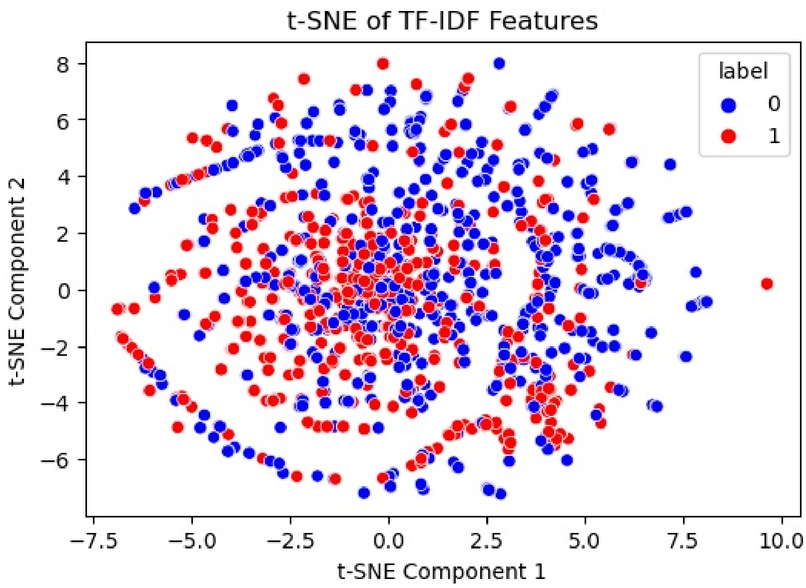
Vectorization, Dimension analysis and visualization

In [7]:
```
# TF-IDF Vectorization
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
X_tfidf = tfidf.fit_transform(df_sample['clean_text'])
```

In [8]:
```
# Custom color palette with solid colors for the labels
custom_palette = {0: 'blue', 1: 'red'}  # Blue for True news, Red for Fake news
```

```python
# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_tfidf.toarray())

plt.figure(figsize=(6, 4))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=df_sample['label'], palette=custom_palette)
plt.title('PCA of TF-IDF Features')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()

# t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30, n_iter=300)
X_tsne = tsne.fit_transform(X_tfidf.toarray())

plt.figure(figsize=(6, 4))
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=df_sample['label'], palette=custom_palette)
plt.title('t-SNE of TF-IDF Features')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()

# MDS
# mds = MDS(n_components=2, random_state=42)
# X_mds = mds.fit_transform(X_tfidf.toarray())

# plt.figure(figsize=(6, 4))
# sns.scatterplot(x=X_mds[:, 0], y=X_mds[:, 1], hue=df_sample['label'], palette=custom_palette)
# plt.title('MDS of TF-IDF Features')
# plt.xlabel('MDS Component 1')
# plt.ylabel('MDS Component 2')
# plt.show()
```





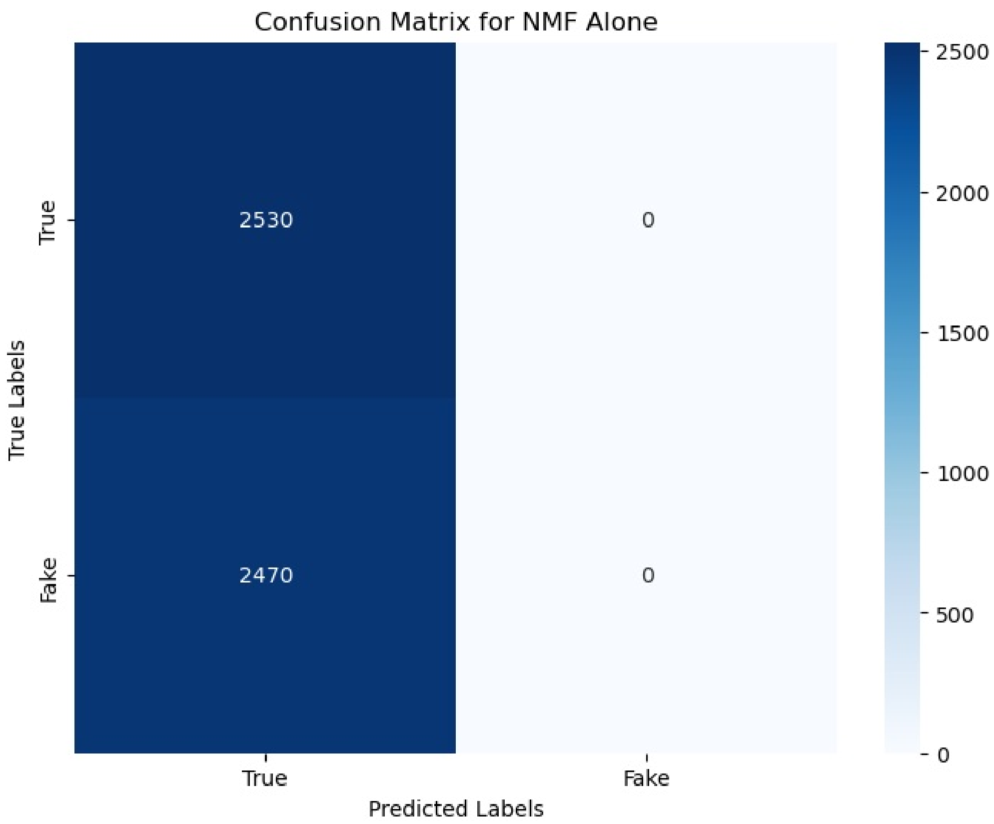
We observe that while PCA reveals some degree of clustering in the data, there is no distinct separation between the clusters. This lack of clear boundaries presents a significant challenge for unsupervised training, as the model must identify decision boundaries without any labeled data. Similarly, the t-SNE analysis shows a comparable pattern, indicating that the hyperplane separating the clusters will be difficult to define, potentially complicating the effectiveness of unsupervised learning methods.

Unsupervised models building, training and evaluation

Non-Negative Matrix Factorization (NMF)

In [9]:

```
# Non-Negative Matrix Factorization (NMF)
nmf = NMF(n_components=2, random_state=42)
X_nmf = nmf.fit_transform(X_tfidf)

# Use the first component of NMF as a basis for classification
# Assuming the first component might represent the "Fake" news category
# Binarize the first component based on a threshold to classify
# (You may need to experiment with this method, as NMF components do not directly correspond to class labels)

# For simplicity, we can classify based on which component is larger
# This assumes that one component is dominant for one class and the other for the other class
binarizer = Binarizer(threshold=0.5)  # You can adjust the threshold as needed
predicted_labels = binarizer.fit_transform(X_nmf[:, 0].reshape(-1, 1)).astype(int).flatten()

# Confusion Matrix
cm = confusion_matrix(df_sample['label'], predicted_labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['True', 'Fake'], yticklabels=['True', 'Fake'])
plt.title('Confusion Matrix for NMF Alone')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Classification Report
report = classification_report(df_sample['label'], predicted_labels, target_names=['True', 'Fake'])
print("Classification Report for NMF Alone:")
print(report)

# Accuracy
accuracy = accuracy_score(df_sample['label'], predicted_labels)
print(f"Accuracy for NMF Alone: {accuracy:.4f}")

# RMSE
rmse = np.sqrt(mean_squared_error(df_sample['label'], predicted_labels))
print(f"RMSE for NMF Alone: {rmse:.4f}")
```



```
Classification Report for NMF Alone:
              precision    recall  f1-score   support

        True       0.51      1.00      0.67      2530
        Fake       0.00      0.00      0.00      2470

    accuracy                           0.51      5000
   macro avg       0.25      0.50      0.34      5000
weighted avg       0.26      0.51      0.34      5000


Accuracy for NMF Alone: 0.5060
RMSE for NMF Alone: 0.7029
```

Optimized NMF

Given the previous results, in this section, we will explore the potential improvement of the model by adjusting the binarizer's threshold. Although this approach involves using the labels, making it not strictly unsupervised, it allows us to assess how much the model could improve if there is a latent feature within the NMF that we are not fully capturing. This experiment aims to determine whether fine-tuning the threshold can enhance the model's ability to classify the data more effectively.

In [14]:

```python
# Custom Transformer for Binarizer threshold tuning
class BinarizerThreshold(BaseEstimator, TransformerMixin):
    def __init__(self, threshold=0.0):
        self.threshold = threshold
        self.binarizer = Binarizer(threshold=self.threshold)

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

    def transform(self, X):
        return self.binarizer.transform(X)

# Non-Negative Matrix Factorization (NMF)
nmf = NMF(n_components=2, random_state=42)
X_nmf = nmf.fit_transform(X_tfidf)

# Create a pipeline to include NMF and the custom binarizer
pipeline = Pipeline([
    ('nmf', nmf),
    ('binarizer', BinarizerThreshold())
])

# Define the parameter grid for the Binarizer threshold
param_grid = {
    'binarizer__threshold': np.arange(0.0, 0.2, 0.01)  # Try different thresholds from 0.0 to 0.2
}

# Set up GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', verbose=2)

# Fit the model and perform grid search on the threshold
grid_search.fit(X_nmf[:, 0].reshape(-1, 1), df_sample['label'])

# Best parameters
print("Best threshold found by GridSearchCV:")
print(grid_search.best_params_)
```

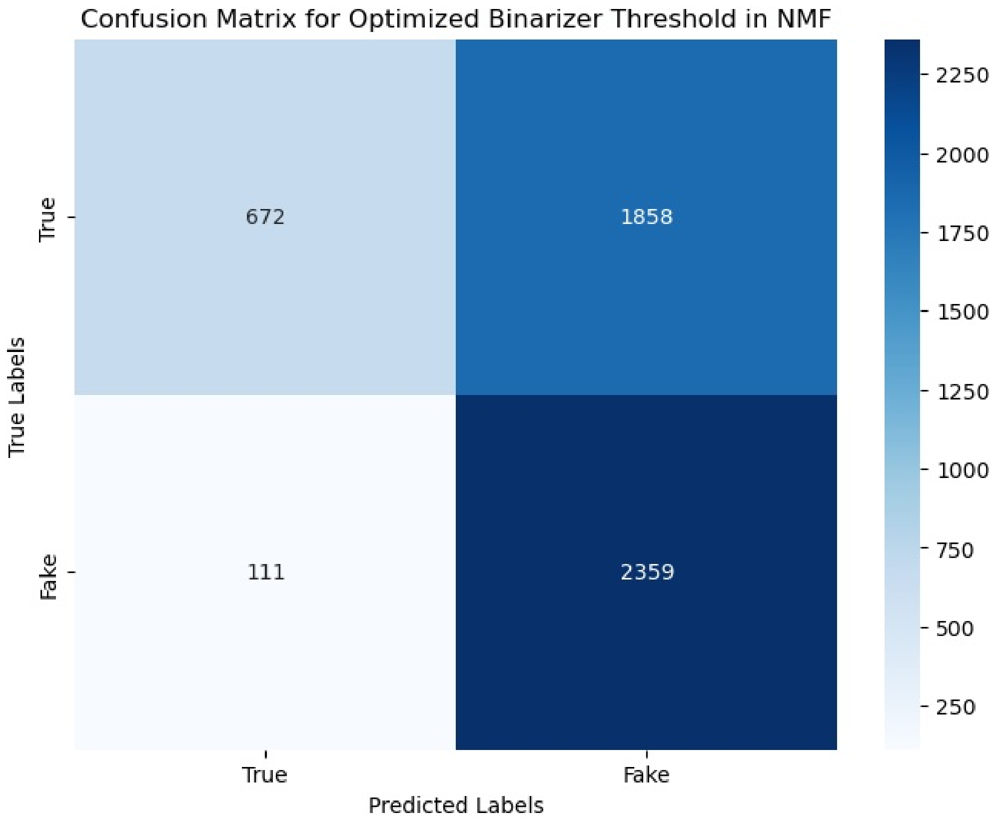
```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] END ...........................binarizer__threshold=0.0; total time=   0.0s
[CV] END ...........................binarizer__threshold=0.0; total time=   0.0s
[CV] END ...........................binarizer__threshold=0.0; total time=   0.0s
[CV] END ...........................binarizer__threshold=0.0; total time=   0.0s
[CV] END ...........................binarizer__threshold=0.0; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.01; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.01; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.01; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.01; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.01; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.02; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.02; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.02; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.02; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.02; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.03; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.03; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.03; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.03; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.03; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.04; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.04; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.04; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.04; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.04; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.05; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.05; total time=   0.0s
```

```
[CV] END .........................binarizer__threshold=0.05; total time=   0.0s
[CV] END .........................binarizer__threshold=0.05; total time=   0.0s
[CV] END .........................binarizer__threshold=0.05; total time=   0.0s
[CV] END .........................binarizer__threshold=0.06; total time=   0.0s
[CV] END .........................binarizer__threshold=0.06; total time=   0.0s
[CV] END .........................binarizer__threshold=0.06; total time=   0.0s
[CV] END .........................binarizer__threshold=0.06; total time=   0.0s
[CV] END .........................binarizer__threshold=0.06; total time=   0.0s
[CV] END .........................binarizer__threshold=0.07; total time=   0.0s
[CV] END .........................binarizer__threshold=0.07; total time=   0.0s
[CV] END .........................binarizer__threshold=0.07; total time=   0.0s
[CV] END .........................binarizer__threshold=0.07; total time=   0.0s
[CV] END .........................binarizer__threshold=0.07; total time=   0.0s
[CV] END .........................binarizer__threshold=0.08; total time=   0.0s
[CV] END .........................binarizer__threshold=0.08; total time=   0.0s
[CV] END .........................binarizer__threshold=0.08; total time=   0.0s
[CV] END .........................binarizer__threshold=0.08; total time=   0.0s
[CV] END .........................binarizer__threshold=0.08; total time=   0.0s
[CV] END .........................binarizer__threshold=0.09; total time=   0.0s
[CV] END .........................binarizer__threshold=0.09; total time=   0.0s
[CV] END .........................binarizer__threshold=0.09; total time=   0.0s
[CV] END .........................binarizer__threshold=0.09; total time=   0.0s
[CV] END .........................binarizer__threshold=0.09; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.1; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.1; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.1; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.1; total time=   0.0s
[CV] END ..........................binarizer__threshold=0.1; total time=   0.0s
[CV] END .........................binarizer__threshold=0.11; total time=   0.0s
[CV] END .........................binarizer__threshold=0.11; total time=   0.0s
[CV] END .........................binarizer__threshold=0.11; total time=   0.0s
[CV] END .........................binarizer__threshold=0.11; total time=   0.0s
[CV] END .........................binarizer__threshold=0.11; total time=   0.0s
[CV] END .........................binarizer__threshold=0.12; total time=   0.0s
[CV] END .........................binarizer__threshold=0.12; total time=   0.0s
[CV] END .........................binarizer__threshold=0.12; total time=   0.0s
[CV] END .........................binarizer__threshold=0.12; total time=   0.0s
[CV] END .........................binarizer__threshold=0.12; total time=   0.0s
[CV] END .........................binarizer__threshold=0.13; total time=   0.0s
[CV] END .........................binarizer__threshold=0.13; total time=   0.0s
[CV] END .........................binarizer__threshold=0.13; total time=   0.0s
[CV] END .........................binarizer__threshold=0.13; total time=   0.0s
[CV] END .........................binarizer__threshold=0.13; total time=   0.0s
[CV] END .........................binarizer__threshold=0.14; total time=   0.0s
[CV] END .........................binarizer__threshold=0.14; total time=   0.0s
[CV] END .........................binarizer__threshold=0.14; total time=   0.0s
[CV] END .........................binarizer__threshold=0.14; total time=   0.0s
[CV] END .........................binarizer__threshold=0.14; total time=   0.0s
[CV] END .........................binarizer__threshold=0.15; total time=   0.0s
[CV] END .........................binarizer__threshold=0.15; total time=   0.0s
[CV] END .........................binarizer__threshold=0.15; total time=   0.0s
[CV] END .........................binarizer__threshold=0.15; total time=   0.0s
[CV] END .........................binarizer__threshold=0.15; total time=   0.0s
[CV] END .........................binarizer__threshold=0.16; total time=   0.0s
[CV] END .........................binarizer__threshold=0.16; total time=   0.0s
[CV] END .........................binarizer__threshold=0.16; total time=   0.0s
[CV] END .........................binarizer__threshold=0.16; total time=   0.0s
[CV] END .........................binarizer__threshold=0.16; total time=   0.0s
[CV] END .........................binarizer__threshold=0.17; total time=   0.0s
[CV] END .........................binarizer__threshold=0.17; total time=   0.0s
[CV] END .........................binarizer__threshold=0.17; total time=   0.0s
[CV] END .........................binarizer__threshold=0.17; total time=   0.0s
[CV] END .........................binarizer__threshold=0.17; total time=   0.0s
[CV] END .........................binarizer__threshold=0.18; total time=   0.0s
[CV] END .........................binarizer__threshold=0.18; total time=   0.0s
[CV] END .........................binarizer__threshold=0.18; total time=   0.0s
[CV] END .........................binarizer__threshold=0.18; total time=   0.0s
[CV] END .........................binarizer__threshold=0.18; total time=   0.0s
[CV] END .........................binarizer__threshold=0.19; total time=   0.0s
[CV] END .........................binarizer__threshold=0.19; total time=   0.0s
[CV] END .........................binarizer__threshold=0.19; total time=   0.0s
[CV] END .........................binarizer__threshold=0.19; total time=   0.0s
[CV] END .........................binarizer__threshold=0.19; total time=   0.0s
Best threshold found by GridSearchCV:
{'binarizer__threshold': 0.0}
```

In [15]:

```python
# Apply the best threshold
best_threshold = grid_search.best_params_['binarizer__threshold']
binarizer = Binarizer(threshold=best_threshold)
predicted_labels = binarizer.fit_transform(X_nmf[:, 0].reshape(-1, 1)).astype(int).flatten()

# Confusion Matrix
cm = confusion_matrix(df_sample['label'], predicted_labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['True', 'Fake'], yticklabels=['True', 'Fake'])
plt.title('Confusion Matrix for Optimized Binarizer Threshold in NMF')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
# Classification Report
report = classification_report(df_sample['label'], predicted_labels, target_names=['True', 'Fake'])
print("Classification Report for Optimized Binarizer Threshold in NMF:")
print(report)

# Accuracy
accuracy = accuracy_score(df_sample['label'], predicted_labels)
print(f"Accuracy for Optimized Binarizer Threshold in NMF: {accuracy:.4f}")

# RMSE
rmse = np.sqrt(mean_squared_error(df_sample['label'], predicted_labels))
print(f"RMSE for Optimized Binarizer Threshold in NMF: {rmse:.4f}")
```




```
Classification Report for Optimized Binarizer Threshold in NMF:
              precision    recall  f1-score   support

        True       0.86      0.27      0.41      2530
        Fake       0.56      0.96      0.71      2470

    accuracy                           0.61      5000
   macro avg       0.71      0.61      0.56      5000
weighted avg       0.71      0.61      0.55      5000

Accuracy for Optimized Binarizer Threshold in NMF: 0.6062
RMSE for Optimized Binarizer Threshold in NMF: 0.6275
```

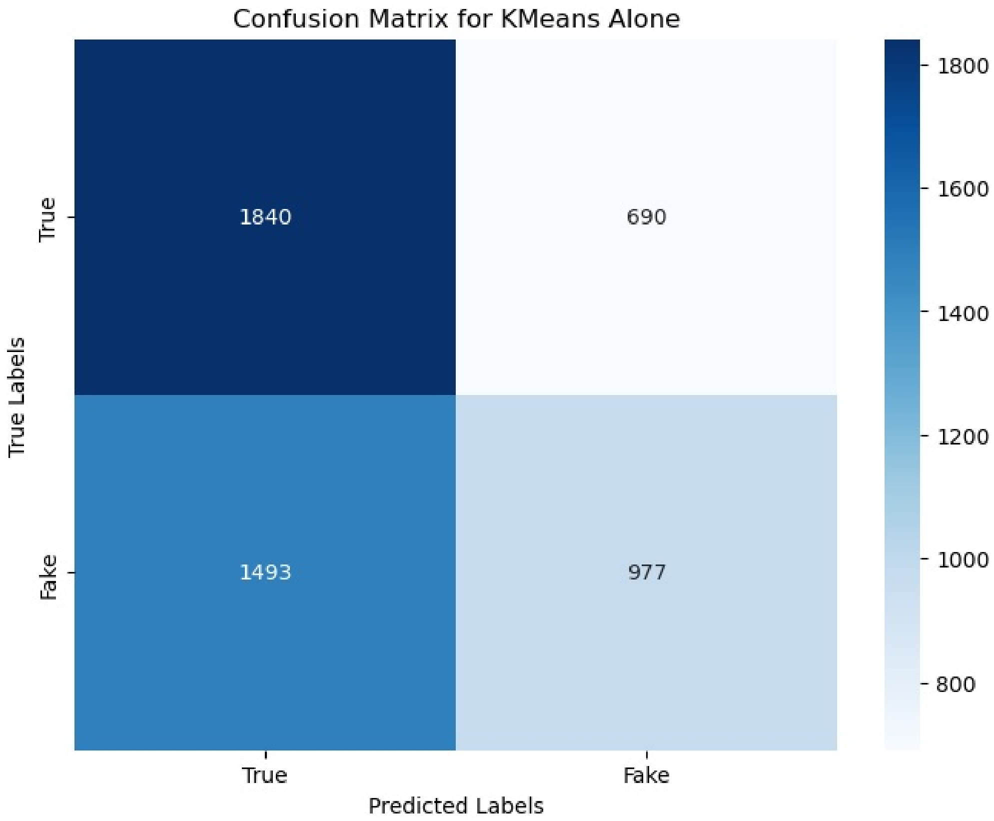
The results from the classification report indicate that optimizing the binarizer threshold in the NMF model did not significantly improve overall accuracy, which remains at approximately 60.62%. However, a notable outcome is the high recall for the "Fake" news class, reaching 96%. This suggests that the model is highly effective at identifying fake news when it is present, though it struggles with correctly identifying true news, as indicated by the low recall of 27% for the "True" class.

The high recall for fake news means that the model is able to detect a large majority of fake news articles, making it useful in scenarios where the primary concern is to minimize false negatives—i.e., missing instances of fake news. However, this comes at the cost of a higher false positive rate, where many true news articles may be incorrectly classified as fake. This trade-off highlights the model's bias towards detecting fake news, potentially due to latent features captured by NMF that are more indicative of the "Fake" class.

K means clustering

In [13]:
```
# KMeans Clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X_tfidf)

# Since KMeans clustering does not directly map to our labels, we need to adjust the predicted labels
# One way to do this is to determine which cluster corresponds to which label by comparing to the true labels

# Create a mapping based on which label is dominant in each cluster
# Get the true labels for each cluster
true_labels_cluster_0 = df_sample['label'][kmeans_labels == 0].mode()[0]
true_labels_cluster_1 = df_sample['label'][kmeans_labels == 1].mode()[0]
```

```python
# Create a mapping for the predicted labels
label_mapping = {0: true_labels_cluster_0, 1: true_labels_cluster_1}

# Apply the mapping
mapped_labels = np.vectorize(label_mapping.get)(kmeans_labels)

# Confusion Matrix
cm = confusion_matrix(df_sample['label'], mapped_labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['True', 'Fake'], yticklabels=['True', 'Fake'])
plt.title('Confusion Matrix for KMeans Alone')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Classification Report
report = classification_report(df_sample['label'], mapped_labels, target_names=['True', 'Fake'])
print("Classification Report for KMeans Alone:")
print(report)

# Accuracy
accuracy = accuracy_score(df_sample['label'], mapped_labels)
print(f"Accuracy for KMeans Alone: {accuracy:.4f}")

# RMSE
rmse = np.sqrt(mean_squared_error(df_sample['label'], mapped_labels))
print(f"RMSE for KMeans Alone: {rmse:.4f}")
```




```
Classification Report for KMeans Alone:
              precision    recall  f1-score   support

        True       0.55      0.73      0.63      2530
        Fake       0.59      0.40      0.47      2470

    accuracy                           0.56      5000
   macro avg       0.57      0.56      0.55      5000
weighted avg       0.57      0.56      0.55      5000

Accuracy for KMeans Alone: 0.5634
RMSE for KMeans Alone: 0.6608
```

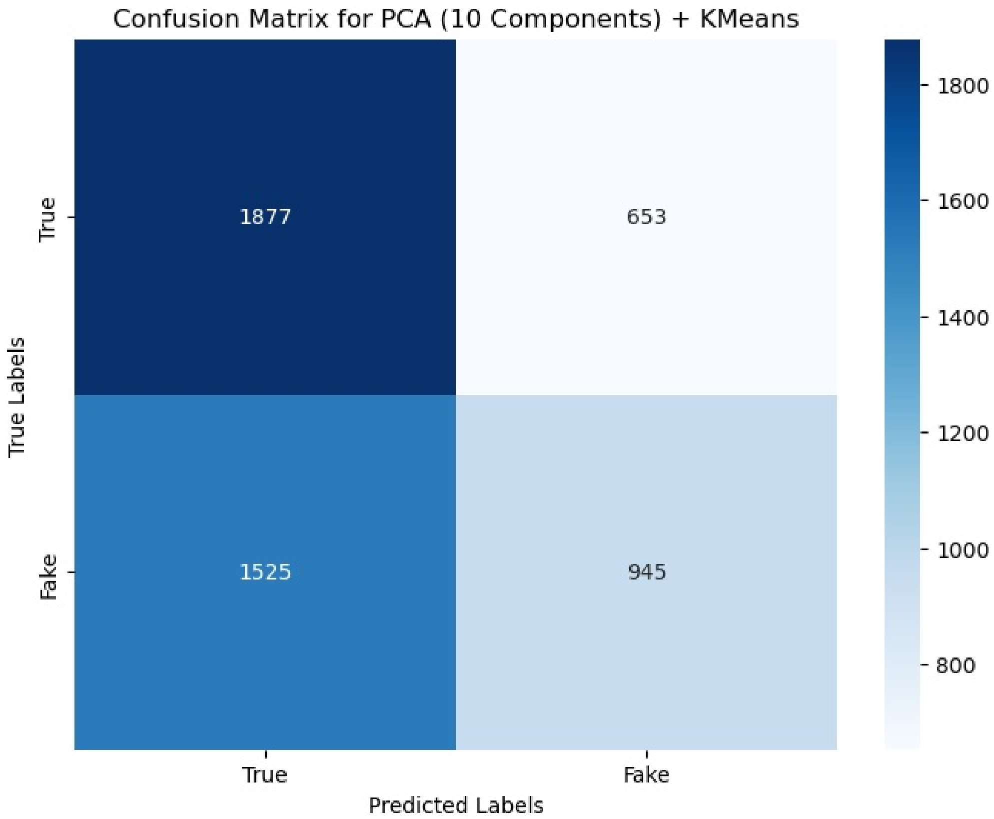
The results for the KMeans clustering model show a modest overall accuracy of 56.34%, with an RMSE of 0.6608. The model exhibits a reasonable balance in precision between the "True" (55%) and "Fake" (59%) news classes, but the recall scores tell a different story.

The recall for "True" news is relatively high at 73%, indicating that the model is better at correctly identifying true news articles. However, the recall for "Fake" news is only 40%, meaning the model frequently fails to identify fake news, leading to a significant number of false negatives.

The F1-scores for both classes are moderate, with "True" news achieving a score of 63% and "Fake" news 47%. These results suggest that while KMeans can somewhat distinguish between true and fake news, its ability to correctly identify fake news is limited, leading to lower overall performance. This is likely due to the unsupervised nature of KMeans, which struggles to form well-separated clusters without labeled data to guide the learning process.

Other combined unsupervised models

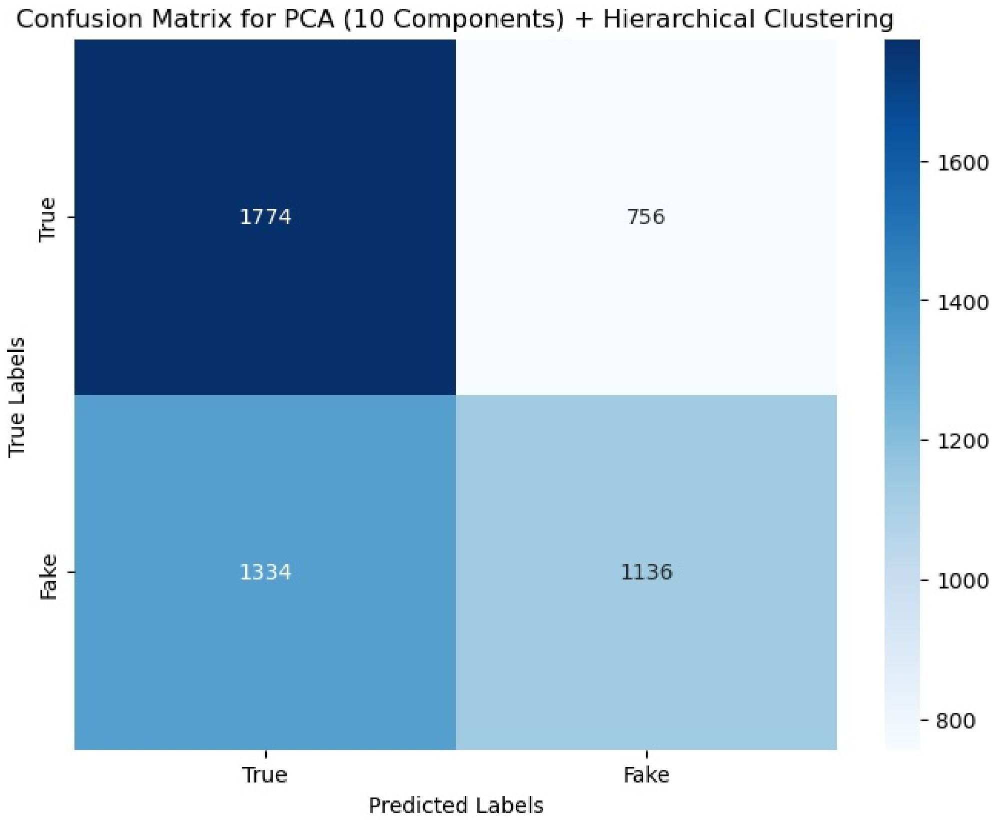
In [16]:
```
# PCA with 10 components
pca = PCA(n_components=10, random_state=42)
X_pca = pca.fit_transform(X_tfidf.toarray())

# Apply KMeans clustering to the PCA-reduced data
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X_pca)

# Since KMeans doesn't know the true labels, we may need to map the clusters to the true labels
# Determine which cluster corresponds to which true label
# This is done by checking which true label is most common in each cluster
cluster_to_label_mapping = {}
for i in range(2):
    mask = (kmeans_labels == i)
    most_common_label = np.bincount(df_sample['label'][mask].astype(int)).argmax()
    cluster_to_label_mapping[i] = most_common_label

# Map the KMeans labels to the true labels
mapped_labels = np.vectorize(cluster_to_label_mapping.get)(kmeans_labels)

# Confusion Matrix for PCA + KMeans model
plt.figure(figsize=(8, 6))
cm_pca_kmeans = confusion_matrix(df_sample['label'], mapped_labels)
sns.heatmap(cm_pca_kmeans, annot=True, fmt='d', cmap='Blues', xticklabels=['True', 'Fake'], yticklabels=['True'
plt.title('Confusion Matrix for PCA (10 Components) + KMeans')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Classification Report for PCA + KMeans model
print("Classification Report for PCA (10 Components) + KMeans:")
print(classification_report(df_sample['label'], mapped_labels))

# Accuracy and RMSE for PCA + KMeans model
accuracy_pca_kmeans = accuracy_score(df_sample['label'], mapped_labels)
rmse_pca_kmeans = np.sqrt(mean_squared_error(df_sample['label'], mapped_labels))
print(f"Accuracy for PCA (10 Components) + KMeans: {accuracy_pca_kmeans:.4f}")
print(f"RMSE for PCA (10 Components) + KMeans: {rmse_pca_kmeans:.4f}")
```



```
Classification Report for PCA (10 Components) + KMeans:
              precision    recall  f1-score   support

           0       0.55      0.74      0.63      2530
           1       0.59      0.38      0.46      2470

    accuracy                           0.56      5000
   macro avg       0.57      0.56      0.55      5000
weighted avg       0.57      0.56      0.55      5000

Accuracy for PCA (10 Components) + KMeans: 0.5644
RMSE for PCA (10 Components) + KMeans: 0.6600
```

In [17]:

```python
# PCA with 10 components
pca = PCA(n_components=10, random_state=42)
X_pca = pca.fit_transform(X_tfidf.toarray())

# Apply Hierarchical Clustering (Agglomerative Clustering) to the PCA-reduced data
hierarchical = AgglomerativeClustering(n_clusters=2)
hierarchical_labels = hierarchical.fit_predict(X_pca)

# Map clusters to the actual labels
# Determine which cluster corresponds to which true label
cluster_to_label_mapping = {}
for i in range(2):
    mask = (hierarchical_labels == i)
    most_common_label = np.bincount(df_sample['label'][mask].astype(int)).argmax()
    cluster_to_label_mapping[i] = most_common_label

# Map the hierarchical clustering labels to the true labels
mapped_labels = np.vectorize(cluster_to_label_mapping.get)(hierarchical_labels)

# Confusion Matrix for PCA + Hierarchical Clustering model
plt.figure(figsize=(8, 6))
cm_pca_hierarchical = confusion_matrix(df_sample['label'], mapped_labels)
sns.heatmap(cm_pca_hierarchical, annot=True, fmt='d', cmap='Blues', xticklabels=['True', 'Fake'], yticklabels=[
plt.title('Confusion Matrix for PCA (10 Components) + Hierarchical Clustering')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Classification Report for PCA + Hierarchical Clustering model
print("Classification Report for PCA (10 Components) + Hierarchical Clustering:")
print(classification_report(df_sample['label'], mapped_labels))

# Accuracy and RMSE for PCA + Hierarchical Clustering model
accuracy_pca_hierarchical = accuracy_score(df_sample['label'], mapped_labels)
rmse_pca_hierarchical = np.sqrt(mean_squared_error(df_sample['label'], mapped_labels))
print(f"Accuracy for PCA (10 Components) + Hierarchical Clustering: {accuracy_pca_hierarchical:.4f}")
print(f"RMSE for PCA (10 Components) + Hierarchical Clustering: {rmse_pca_hierarchical:.4f}")
```



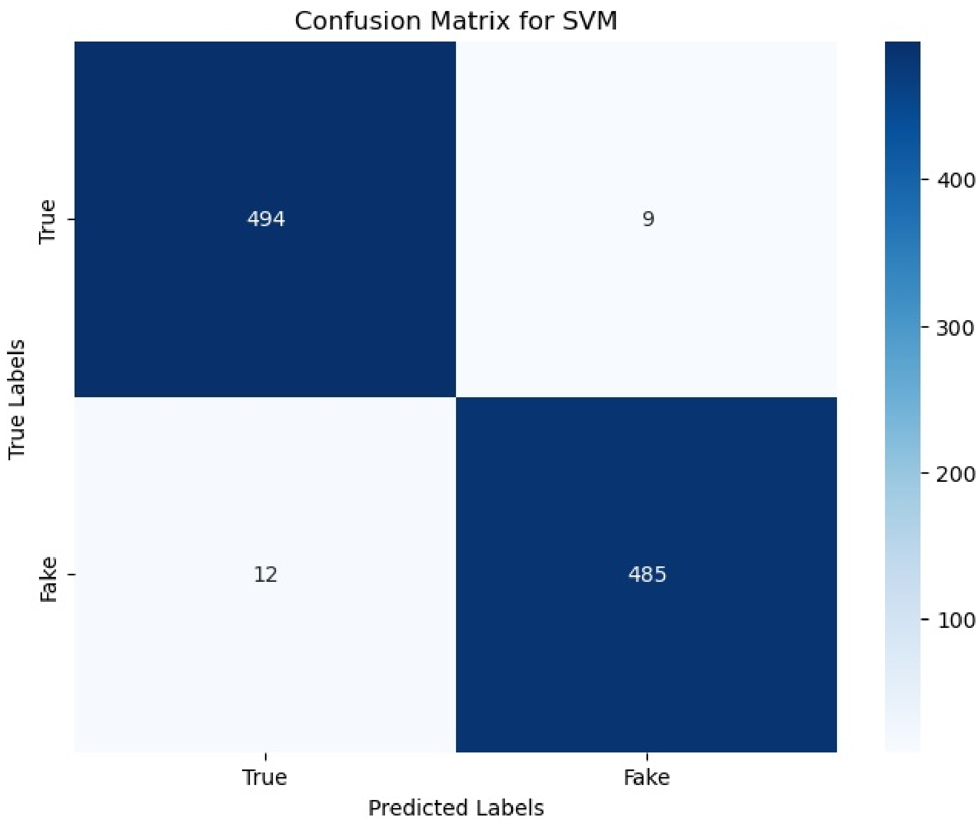
```
Classification Report for PCA (10 Components) + Hierarchical Clustering:
              precision    recall  f1-score   support


           0       0.57      0.70      0.63      2530
           1       0.60      0.46      0.52      2470

    accuracy                           0.58      5000
   macro avg       0.59      0.58      0.58      5000
weighted avg       0.59      0.58      0.58      5000

Accuracy for PCA (10 Components) + Hierarchical Clustering: 0.5820
RMSE for PCA (10 Components) + Hierarchical Clustering: 0.6465
```

Supervised models building, training and evaluation

Support Vector Machine

In [18]:

```python
# Classification using Support Vector Machines (SVM)
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, df_sample['label'], test_size=0.2, random_state=42

# Initial SVM model
svm = SVC()
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)

# Confusion Matrix for initial SVM model
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred_svm)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['True', 'Fake'], yticklabels=['True', 'Fake'])
plt.title('Confusion Matrix for SVM')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Classification Report for initial SVM model
print("Classification Report for SVM:")
print(classification_report(y_test, y_pred_svm))

# Accuracy and RMSE for initial SVM model
accuracy = accuracy_score(y_test, y_pred_svm)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_svm))
print(f"Accuracy for SVM: {accuracy:.4f}")
print(f"RMSE for SVM: {rmse:.4f}")
```

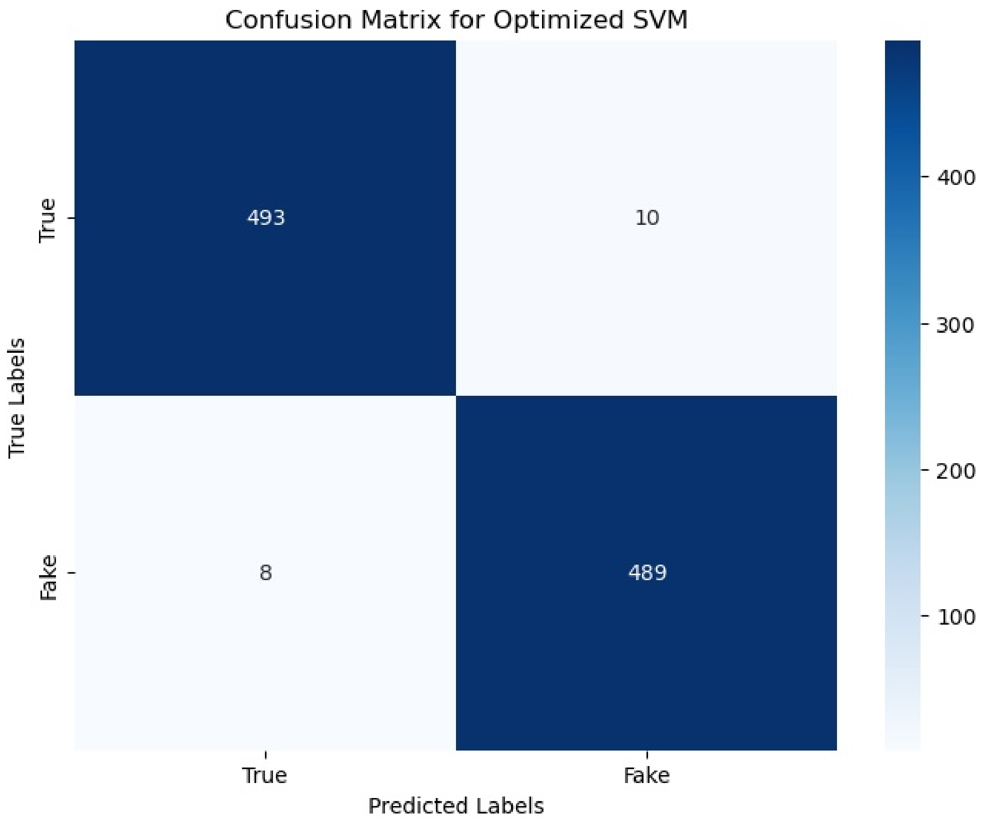


```
Classification Report for SVM:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       503
           1       0.98      0.98      0.98       497

    accuracy                           0.98      1000

   macro avg       0.98      0.98      0.98      1000
weighted avg       0.98      0.98      0.98      1000

Accuracy for SVM: 0.9790
RMSE for SVM: 0.1449
```

Optimized Support Vector Machine

In [19]:

```python
# Hyperparameter Optimization for SVM
svm_param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
svm_grid_search = GridSearchCV(SVC(), svm_param_grid, cv=5, scoring='accuracy')
svm_grid_search.fit(X_train, y_train)
best_svm = svm_grid_search.best_estimator_

# Predictions using the optimized SVM model
y_pred_best_svm = best_svm.predict(X_test)

# Confusion Matrix for optimized SVM model
plt.figure(figsize=(8, 6))
cm_best = confusion_matrix(y_test, y_pred_best_svm)
sns.heatmap(cm_best, annot=True, fmt='d', cmap='Blues', xticklabels=['True', 'Fake'], yticklabels=['True', 'Fak
plt.title('Confusion Matrix for Optimized SVM')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Classification Report for optimized SVM model
print("Classification Report for Optimized SVM:")
print(classification_report(y_test, y_pred_best_svm))

# Accuracy and RMSE for optimized SVM model
accuracy_best = accuracy_score(y_test, y_pred_best_svm)
rmse_best = np.sqrt(mean_squared_error(y_test, y_pred_best_svm))
print(f"Accuracy for Optimized SVM: {accuracy_best:.4f}")
print(f"RMSE for Optimized SVM: {rmse_best:.4f}")

# Display the best parameters found by GridSearchCV
print("Best SVM Model Parameters:")
print(best_svm)
```



```
Classification Report for Optimized SVM:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       503
           1       0.98      0.98      0.98       497

    accuracy                           0.98      1000
   macro avg       0.98      0.98      0.98      1000
weighted avg       0.98      0.98      0.98      1000

Accuracy for Optimized SVM: 0.9820
RMSE for Optimized SVM: 0.1342
Best SVM Model Parameters:
SVC(C=10, kernel='linear')
```

The results for both the standard SVM and the Optimized SVM models demonstrate exceptionally high performance in classifying the data, with accuracy scores of 97.90% and 98.20%, respectively. Both models achieve near-perfect precision, recall, and F1-scores across the "True" and "Fake" news classes, indicating their ability to correctly classify almost all instances in the dataset.

The RMSE values are low, with the optimized SVM model slightly outperforming the standard SVM model (0.1342 vs. 0.1449), which suggests that the optimization process, particularly with the parameters (C=10, kernel='linear'), has marginally improved the model's performance.

Overall, these results highlight the effectiveness of supervised learning approaches, particularly SVM, in accurately detecting and classifying fake news when trained on labeled data. The minimal difference between the standard and optimized models indicates that SVM, even without extensive tuning, is highly capable in this context.

Results Summary

Method	Accuracy	RMSE
NMF Alone	0.5060	0.7029
Optimized Binarizer Threshold in NMF	0.6062	0.6275
KMeans Alone	0.5634	0.6608
PCA (10 Components) + KMeans	0.5644	0.6600
PCA (10 Components) + Hierarchical Clustering	0.5820	0.6465
SVM	0.9790	0.1449
Optimized SVM	0.9820	0.1342

Discussion

The primary objective of this project was to explore the effectiveness of unsupervised machine learning models in detecting and classifying fake news. However the results indicate that while unsupervised models like NMF, KMeans, and hierarchical clustering can offer some degree of differentiation between true and fake news, their performance falls short when compared to supervised models. The highest accuracy among the unsupervised approaches was achieved by the optimized binarizer threshold in NMF, with an accuracy of 0.60. This suggests that while the model was somewhat effective in identifying patterns within the data, the lack of labeled guidance limited its overall performance.

In contrast, supervised models, demonstrated significantly higher accuracy, with the standard SVM achieving 0.97 without any tunning. The substantial gap between the performance of unsupervised and supervised models underscores the importance of labeled data in training accurate classification models. The supervised models were able to leverage the labeled data to fine-tune decision boundaries effectively, resulting in near-perfect classification.

The results clearly indicate that supervised learning models, when provided with labeled data, vastly outperform unsupervised approaches in the context of fake news detection. The unsupervised models struggled to form well-defined clusters or decision boundaries, likely due to the complexity of the text data and the subtle differences between true and fake news that require more sophisticated feature extraction.

The features derived from text data using unsupervised methods may not have been sufficiently distinctive to separate true and fake news effectively. Fake news detection is inherently complex and may require more nuanced understanding and features that unsupervised models struggle to capture without labeled data.

This may need more advanced feature extraction engineering, such those deerived form NLP algorithms like BERT and combining techniques to combine strengths of different model. Nevertheless, with this project we got valuable insights into the challenges of unsupervised learning.

In [ ]: