A Comparative Analysis of Machine Learning Algorithms for Fake News Detection

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Karthik Cherukuru (AP23110010658)

Gubba Yasodhar (AP23110010512)

Chitirala Naga Vasanth Kumar (AP23110011121)

Janjanam Jaswanth (AP23110010897)

Under the Guidance of

Dr. Abhijit Dasgupta

Assistant Professor, Department of Computer Science and Engineering

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[July, 2025]

Certificate

Date: _____

This is to certify that the work present in this Project entitled "**A Comparative Analysis of Machine Learning Algorithms for Fake News Detection**" has been carried out by **Karthik Cherukuru, Gubba Yasodhar, Chitirala Naga Vasanth Kumar, Janjanam Jaswanth** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. **Dr. Abhijit Dasgupta**

Assistant Professor

Department of Computer Science & Engineering

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Dr. Abhijit Dasgupta**, Assistant Professor ,Department of Computer Science & Engineering, SRM University, Andhra Pradesh, for his kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

Karthik Cherukuru (AP23110010658)

Gubba Yasodhar (AP23110010512)

Chitirala Naga Vasanth Kumar (AP23110011121)

Janjanam Jaswanth (AP23110010897)

Abstract

The problem of the rapid spread of digital falsities or fake news spells great danger to social communication and democracy. In this project the fake news detection is seen as a binary text classification issue, and the news articles are programmatically labeled as either 'REAL' or 'FAKE'. The overall goal is to install, test, and analyze the five different algorithms of machine learning with the idea of which is the most effective model as per predictive performance and cost of computation.

The analysis will make use of a balanced sample of 6,335 news items with a ratio of 50 percent of Fake and 50 percent of Real news items. An entire Natural Language Processing (NLP) pipeline was designed to process the text data to be ready to model. The processes that were done in this pipeline included the combination of title and article text, normalizing cases text, eliminating punctuation and special characters, tokenization, and eliminating stop wording. Word normalization was done using lemmatization instead of stemming since it also retains semantic meaning and feature extraction was done using Term Frequency-Inverse Document Frequency (TF-IDF) because they are intelligent and give weights to the words accordingly in spite of relative importance.

In the heart of this project lies a particular comparison of 5 supervised learning algorithms namely Multinomial Naive Bayes, K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machine (SVM) and a Feedforward Neural Network (FFNN). Projected algorithms were trained and tested and their performance measured on accuracy, precision, recall, F1-score and training and prediction computation time.

The findings suggest that there was a well hierarchical performance. The Support Vector Machine recorded the best F1-Score of the two classes: 0.93 (second best score was the Feedforward Neural Network and the Logistic Regression which yielded a score of 0.92). Multinomial Naive Bayes provided a reasonable baseline of the F1-score 0.88, whereas K-Nearest Neighbors was behind at 0.87 with hindrances due to the high-dimensionality of the text data.

An important parameter of the analysis was its computing effectiveness. Although KNN was quite quick in regards to the training time (memorizing the data), its classification time was very poor (4.33 seconds) and therefore not much use in real-life practice. On the other hand, such

models as Multinomial Naive Bayes and Logistic Regression were unbelievably quick, taking only 0.02 seconds to make predictions. Although the accuracy of the FFNN and SVM are very high, the amount of training time to achieve a tiny improvement in accuracy is long (21.43s and 54.14s respectively) as compared to the time taken by Logistic regression to improve accuracy (2.16s).

It is inferred in this report that based on a comprehensive consideration of the following two aspects of predictive accuracy and computational cost that it can be taken that:

The most effective algorithm in this fake news detection task is the **Logistic Regression**. It offers the ideal trade-off, giving excellent predictive results (92% F1-score) with no significant overhead to train and make predictions, making it a very consistently high-achieving and therefore realistic choice. Future investigations may try cross-dataset validation, a real-time detection system, and the implementation of more complex neural network models (such as LSTMs and Transformers).

Table of Contents

1. Introduction to Automated Fake News Detection

1.1. The Problem of Digital Misinformation

The circulation of misinformation, also known as fake news, is a major problem in the modern digital environment that challenges the discourse in the society, the trust of the masses, and the democratic styles. The readiness and the time it takes to spread misleading information or information that cannot be verified by using online platforms requires the creation of strong and automated detection technologies. This project tackles the issue of misinformation by posing it on the perspective of supervised machine learning: text classification (binary). One is to programmatically examine the contents of a news article and to provide one of two pre-determined labels that a news item can be, 'REAL' or 'FAKE'. With the application of Natural Language Processing (NLP) and machine learning, it is possible to look at linguistic patterns, style or content features that can be used to identify whether a journalism item is fantastic or not.

1.2. Project Objectives and Scope

The main goal of the project will be to apply, critically test, and qualitatively compare five different machine learning algorithms to solve the problem of fake news spotting on a given set of news articles on a given dataset. The chosen algorithms cover fairly broad spectrum of modeling paradigms:

1. **Multinomial Naive Bayes:** A classic probabilistic classifier.
2. **K-Nearest Neighbors (KNN):** An instance-based, non-parametric method.

3. **Logistic Regression:** A powerful and interpretable linear model.
4. **Support Vector Machine (SVM):** A maximum-margin linear classifier.
5. **Feedforward Neural Network (FFNN):** A foundational deep learning model.

The second, but important task is to determine the algorithm that exhibits the "optimal effectiveness." Efficiency will be a composite, meaning that it will be measured both on the performance of a predictive task, i.e., accuracy, precision, recall, and F1-score, and on time overhead, i.e., the time taken to train the model and make a prediction.

This report has been organized in such a way that the reader can be taken through the project lifecycle. Section 2 contains the description of the dataset and the full preprocessing pipeline of the text that was developed to convert the raw text into the machine-readable format. Section 3 contains a theoretical description, as well as practical application of each of the five algorithms with their respective performance outputs. Section 4 then provides a comparative standard of all the models evaluating their relative strengths and weaknesses of all the models based on accuracy and the cost of computation. Lastly, Section 5 gives a concluding statement regarding this endeavour by giving a verdict on the most effective algorithm to use in this undertaking and gives a recommendation of what should be done to the project in the future.

2. Data Foundation and Text Preprocessing

2.1. Dataset Exploration and Initial Analysis

The news.csv data is a news article dataset that has been labelled as either FAKE or REAL. The data set is in three major columns:

title, text and label. A first glance at the data shows a clean and well-structured corpus appropriate to a task of classification.

The database encompasses the 6,335 separate news publications. One of the severe traits of this dataset is its class balance. There are 3,171 articles in the category of FAKE and 3,164 in the category of REAL. This almost ideal 50/50 balance is very beneficial, since it prevents the possibility of a model becoming biased against a majority class. As a result, the common performance metrics such as accuracy can be applied as a sufficient source of information on the effectiveness of the model without any specific methods to handle class imbalances. Additionally, an integrity check shows that no field has any missing or null values, which makes the issues of data loading and cleaning of the project simplified at the initial stages.

2.2. The Natural Language Processing (NLP) Pipeline

Raw text data is never structured, that is why it cannot be directly inputted into machine learning algorithms. It demands a sequence of changes in order to change it into clean, standardized, and numeric form. Such a multi-step process can be done as an NLP pipeline. The pipeline proposed in this project has a number of consecutive steps with particular objectives to process and prepare the data to further feature extraction.

Step 1: Text Combination

Both the headline and the content of a news report makes a good source of information. The title usually gives a brief summary or a hook on the sensational side and then the body has the detailed story. These two columns are then converted into one text field to generate as many features as possible of every document. This will make sure that the linguistic patterns of the headline and article body are present to the models.

Step 2: Case Normalization

Words in a text data may be stored in all forms (e.g., Election, election, ELECTION). Lower case conversion of all text is to be made so that the machine learning model treats these as the same word. It is

a primary standardization procedure and serves to eliminate the vocabulary being artificially enlarged with duplicating tokens.

Step 3: Punctuation and Special Character Removal

Special characters, such as punctuations (e.g.,., ', ', '!'), numbers, and other types of common characters usually do not add much semantic value to this classification exercise, and could be treated as noise. Regular expressions are then used to methodically strip the non-alphabetics out of the text, leaving the words and whitespace only.

Step 4: Tokenization

The conversion of the strings of text into a list of individual words, or tokens is called tokenization. Such tokens are the foundation of constructing the vocabulary and feature vectors into the machine learning models.

Step 5: Stop Word Removal

The languages are full of common words that are syntactically indispensable and have less description like the, a, in and is. These are called the stop words. This is an essential step to diminish the dimensionality of the data and to minimize the computational load and focus the models on the words that better qualify to be informative regarding the distinction between the two classes of news, namely, between the FAKE and the REAL ones. The stop word list of the standard English that is built into the Natural Language Toolkit (NLTK) library is employed.

2.3. Word Normalization: Stemming vs. Lemmatization

Before application of stop removal and some other method of removing tokens, the second thing to do is to normalize the words, which should be to minimize various forms of a word to a simple base form.10 For instance, the phrases run, runs and running should be normalized to one word. This can be done in two main ways by stemming and lemmatization.

- **Stemming:** This method applies a blunt, rule-based heuristic method of lopping the ends of words. As an example, a Porter stemmer would turn the words studies, studying and study into the shared stem “studi”. Although generally computationally rapid, the key disadvantage with this process is that there are many scenarios where stem produced after conducting such a process may not represent an actual word and hence the meaning of the

word is lost.

- **Lemmatization:** It is a more cultured and linguistically conscious process. It has access to a dictionary (such as WordNet) and takes into account the part-of-speech of a word in order to obtain its right dictionary base form which is called lemma. As in the case of, to put an example, it correctly dwarfs down the instances of both studies and studying itself to the lemma of study. It can also rightly say that the lemma of better is good.

In this project, **lemmatization was chosen** as the strategy of normalization. Though it is computationally demanding compared to stemming, it ensures that the semantic meaning of words is well maintained; a fact that is paramount to a subtle task such as fake news detection. Having more products of the lemmatization process which are high-quality and more accurate tokens is likely to establish a more effective featured set, and the extra processing cost is worth it.

2.4. Feature Extraction: Bag-of-Words (BoW) vs. TF-IDF

Representation of the processed lists of word tokens to the numerical vectors that can be used by machine learning models is the last part of data preparation. This has been referred to as feature extraction or vectorization.

- **Bag-of-Words (BoW):** BoW model is the simplest example of the idea where each document is expressed as a word count vector. It makes a vocabulary of all the unique words present in the corpus and it counts the frequency of each word in every document. This approach does not recognize grammar and word order and recognizes the occurrence of words. Its predominant weakness is that it will concentrate each word in an equal measure i.e., the word, election would hold as much inherent weight as another word, article.
- **Term Frequency-Inverse Document Frequency (TF-IDF):** TF-IDF is a more complex statistical factor that gauges the relevance of the word to a document within a repository of

documents (a corpus). It ensures the shortcoming of BoW, as a weight is given to every word. TF-IDF score is a product of two things:

- **Term Frequency (TF):** This is the measure of occurrence of a term in a document. The greater the value the more important the word is to the particular document.
- **Inverse Document Frequency (IDF):** It helps measure the more or less frequent a word is in the whole corpus. Words such as news or say will have a low IDF score and more specific ones such as Comey or Sanders will have a high IDF score.

The multiple that emerges is the TF-IDF score, and that is large in case the words are common in a document but are not prevalent in the whole collection, thus they will be powerful discriminators of the content of the document.

In this work, the **TF-IDF vectorization was selected** as a feature extraction method. Its capability to intelligently weigh words depending on their significance is much more superb compared to the counting system of BoW. This weighting scheme is especially helpful when trying to detect fake news as certain keywords which are less frequently used can be very strong indicators to authenticity of a document. This will be done using the TfidfVectorizer of the scikit-learn library.

The natural order of steps taken including striking texts together, lemmatization, and lastly TF-IDF vectorization is a calculated move towards the quality of data. Individual stages in this cascade of preprocessing are to filter out imperfections in the raw text, diminish the amount of noise, and generate a feature set which is semantically significant and discriminative in terms of numbers. The result of this strong base is learnt to be the direct introduction of greater performance with the various novel forms of classification models that have to be tested.

Code: Data Loading and Full Preprocessing Pipeline

```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import time
```

```
print("Loading dataset...")
try:
    df = pd.read_csv('news.csv', on_bad_lines='skip')
    print("Dataset loaded successfully.")
except Exception as e:
    print(f"Error loading dataset: {e}")
    exit()

# Drop unnamed column if exists
if 'Unnamed: 0' in df.columns:
    df = df.drop('Unnamed: 0', axis=1)

# Combine title and text
df['full_text'] = df['title'] + ' ' + df['text']
print("'full_text' column created.")

if 'label' not in df.columns:
    print("Error: 'label' column not found in the dataset.")
    exit()
```

```
import nltk
nltk.download('all')
```

```
print("Starting text preprocessing...")
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
sentence_tokenizer = nltk.PunktSentenceTokenizer()

def preprocess_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = []
    for sentence in sentence_tokenizer.tokenize(text):
        tokens.extend(nltk.word_tokenize(sentence))
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(lemmatized_tokens)

df['processed_text'] = df['full_text'].apply(preprocess_text)
print("Text preprocessing complete.")
```

```
print("Performing TF-IDF Vectorization and data splitting...")
X = df['processed_text']
y = df['label']

tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_tfidf = tfidf_vectorizer.fit_transform(X).toarray()

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42, stratify=y)

print(f"Shape of training features: {X_train.shape}")
print(f"Shape of testing features: {X_test.shape}")
```

3. Algorithmic Implementation and Performance Analysis

This part explains how they applied and assessed five identified machine learning algorithms. The models are fitted with the preprocessed and vectorized training data (X_train, y_train). and assessed on the unseen test data (X_test y_test). Standard classification metrics will be evaluated (accuracy, precision, recall, F1-score) as well as a confusion matrix of analysis of errors and metrics on training time and prediction time will be calculated.

3.1. Multinomial Naive Bayes: A Probabilistic Baseline

3.1.1. Theoretical Framework

The Naive Bayes family of algorithms and Bayes theorem have their foundation in probability theory due to Bayes theorem. The argument of the naive nature of the algorithm lies in the main assumption made by the algorithm: all features (in the case each score of the TF-IDF value of each word) are conditional independent of each other based on the class label ('FAKE' or 'REAL'). Even though this assumption is unlikely to be true in all real-world texts (the likelihood

that one is a Hillary indeed influences the probability of being a Clinton), the algorithm is very successful in practice, particularly text classification. The Multinomial Naive Bayes is specifically suited to working on a feature that records counts or frequencies, hence its applicability on text data that has been TF-IDF vectorized. By calculating the likelihood that a document belongs to a particular class and reporting the most likely one, Multinomial Naive Bayes returns the classification as the one with the highest probability.

3.1.2. Implementation in Python

The implementation uses the MultinomialNB class from scikit-learn. The model is trained on the TF-IDF vectors and their corresponding labels.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

print("\n--- Multinomial Naive Bayes ---")
start_time = time.time()
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
end_time = time.time()
print("Training Time:", end_time - start_time, "seconds")

start_time = time.time()
y_pred_nb = nb_model.predict(X_test)
end_time = time.time()
print("Prediction Time:", end_time - start_time, "seconds")

print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))

cm_nb = confusion_matrix(y_test, y_pred_nb)
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues', xticklabels=['Fake', 'Real'], yticklabels=['Fake', 'Real'])
plt.title("Confusion Matrix: Naive Bayes")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

3.1.3. Results and Evaluation

- **Accuracy:** 0.8768
- **Training Time:** 0.179424 seconds
- **Prediction Time:** 0.0292 seconds

Classification Report:

	precision	recall	f1-score	support
FAKE	0.86	0.89	0.88	633
REAL	0.89	0.86	0.87	634
accuracy			0.88	1267
macro avg	0.88	0.88	0.88	1267
weighted avg	0.88	0.88	0.88	1267

Confusion Matrix Analysis: The model correctly identified 566 fake news articles and 545 real news articles. However, it misclassified 67 fake articles as real and 89 real articles as fake. The model shows a slight bias toward predicting fake, as reflected in the higher recall for the fake class compared to the real class.

Discussion: Multinomial Naive Bayes gives a robust and very fast baseline, as can be expected. This feature of training within fractions of a second makes it quite effective on initial analysis, but predictive power is a bit less than in more complex models.

3.2. K-Nearest Neighbors (KNN): An Instance-Based Approach

3.2.1. Theoretical Framework

K-Nearest Neighbors is an instance based algorithm which is non-parametric. In contrast to other models, whereby the approach teaches a function based on the training data, KNN only memorizes the training dataset. It is thus commonly referred to as a "lazy learner". KNN finds

different classification categories to which a set of data points belong by determining the nearest neighbors of that individual within the training set using a distance measure, usually Euclidean distance. The new data point would be labeled the most common class of its neighbors nearest to the decision boundary; a number of neighbors (sometimes called k-Nearest neighbors) called k is an important hyper-parameter; a small k will potentially sensitise the model to noise in the data, and a large k can be very slow and potentially oversmooth the solution.

3.2.2. Implementation in Python

The implementation uses scikit-learn's KNeighborsClassifier. A value of k=5 is chosen as a common starting point.

```
from sklearn.neighbors import KNeighborsClassifier

print("\n--- K-Nearest Neighbors ---")
start_time = time.time()
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
end_time = time.time()
print("Training Time:", end_time - start_time, "seconds")

start_time = time.time()
y_pred_knn = knn_model.predict(X_test)
end_time = time.time()
print("Prediction Time:", end_time - start_time, "seconds")

print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Classification Report:\n", classification_report(y_test, y_pred_knn))

cm_knn = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Blues', xticklabels=['Fake', 'Real'], yticklabels=['Fake', 'Real'])
plt.title("Confusion Matrix: KNN")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

3.2.3. Results and Evaluation

- **Accuracy:** 0.87213
- **Training Time:** 0.02337 seconds
- **Prediction Time:** 4.3398 seconds

Classification Report:

	precision	recall	f1-score	support
FAKE	0.90	0.84	0.87	633
REAL	0.85	0.91	0.88	634
accuracy			0.87	1267
macro avg	0.87	0.87	0.87	1267
weighted avg	0.87	0.87	0.87	1267

Confusion Matrix Analysis: The KNN model correctly identified 529 fake articles and 576 real articles. It exhibits a tendency to misclassify fake news as real, with 104 such instances, while being more effective at identifying real news, with only 58 misclassified as fake. This is reflected in the high recall for the real class and lower recall for the fake class.

Discussion: The performance of the KNN underscores its individual characteristic performance. The process is almost immediate as the model just stores the data. Nevertheless, the forecast step is very slow. The reason is that, in predicting the 1,267 items in the test set, the 1,267 test items in the test set, the model will have to calculate the distance between itself and all 5,068 training items in a 5,000 dimensional space. Such an inference time computational cost, along with its poorer accuracy because of the curse of dimensionality in high-dimensional text data, means that KNN is inefficient to perform this task.

3.3. Logistic Regression: A Powerful Linear Classifier

3.3.1. Theoretical Framework

The algorithm is called Logistic Regression and is not a regression but a classification algorithm. It is a linear model computing the probability of an outcome of interest. This it does by computing a weighted sum of the input features and then going through a logistic function (or sigmoid function, as it is alternatively called). The sigmoid is a real-valued input transforming them into 0-1, which may be seen as a probability, before passing such a probability through a decision threshold (usually 0.5) to make a prediction regarding classification membership in the latter. Logistic Regression is very efficient, interpretable and is frequently a very strong baseline in a text classification problem.

3.3.2. Implementation in Python

The implementation uses scikit-learn's LogisticRegression class.

```
from sklearn.linear_model import LogisticRegression

print("\n--- Logistic Regression ---")
start_time = time.time()
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)
end_time = time.time()
print("Training Time:", end_time - start_time, "seconds")

start_time = time.time()
y_pred_lr = lr_model.predict(X_test)
end_time = time.time()
print("Prediction Time:", end_time - start_time, "seconds")

print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))

cm_lr = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues', xticklabels=['Fake', 'Real'], yticklabels=['Fake', 'Real'])
plt.title("Confusion Matrix: Logistic Regression")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

3.3.3. Results and Evaluation

- **Accuracy:** 0.91475
- **Training Time:** 2.1649 seconds
- **Prediction Time:** 0.0208 seconds

Classification Report:

	precision	recall	f1-score	support
FAKE	0.89	0.94	0.92	633
REAL	0.94	0.89	0.91	634
accuracy			0.91	1267
macro avg	0.92	0.91	0.91	1267
weighted avg	0.92	0.91	0.91	1267

Confusion Matrix Analysis: The Logistic Regression model demonstrates strong and balanced performance. It correctly identified 595 fake articles and 564 real articles, while misclassifying only 38 fake articles as real and 70 real articles as fake. The close alignment of precision, recall, and F1-scores for both classes suggests the model is both accurate and unbiased.

Discussion: Logistic Regression is an efficient and very successful classifier in this task. It has an extremely high accuracy using rapid training and prediction times that take near-zero time, becoming one of the best candidates in terms of algorithm efficiency.

3.4. Support Vector Machine (SVM): Maximizing the Margin

3.4.1. Theoretical Framework

A support vector machine also known as SVM is a strong classifier that is based on supervised learning. The essence of SVM is the selection of an optimal hyperplane that can most appropriately partition the data points of different classes in high dimensional space. The optimal hyperplane is the one which has the largest margin, that is the distance between the hyperplane and nearest data points belonging to the two classes. These closest points become known as support vectors because it is these points that actually support or characterize the decision place of the hyperplane. Support vectors are very attractive when it comes to SVMs, and this is specifically in high-dimensional spaces, say for text classification where the features used can be extremely large (number of words in the vocabulary). In such a case the choice of kernel used by the algorithm is linear implying that the algorithm is trying to find a linear decision boundary.

3.4.2. Implementation in Python

The implementation uses scikit-learn's SVC (Support Vector Classifier) with kernel='linear'.

```
from sklearn.svm import SVC

print("\n--- Support Vector Machine ---")
start_time = time.time()
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
end_time = time.time()
print("Training Time:", end_time - start_time, "seconds")

start_time = time.time()
y_pred_svm = svm_model.predict(X_test)
end_time = time.time()
print("Prediction Time:", end_time - start_time, "seconds")

print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))

cm_svm = confusion_matrix(y_test, y_pred_svm)
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues', xticklabels=['Fake', 'Real'], yticklabels=['Fake', 'Real'])
plt.title("Confusion Matrix: SVM")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

3.4.3. Results and Evaluation

- **Accuracy:** 0.9281
- **Training Time:** 54.1464 seconds
- **Prediction Time:** 9.7623 seconds

Classification Report:

	precision	recall	f1-score	support
FAKE	0.92	0.94	0.93	633
REAL	0.94	0.91	0.93	634
accuracy			0.93	1267
macro avg	0.93	0.93	0.93	1267
weighted avg	0.93	0.93	0.93	1267

Confusion Matrix Analysis: The SVM model achieves strong and balanced performance. It correctly classified 596 fake articles and 580 real articles, while misclassifying 37 fake articles as real and 54 real articles as fake. Compared to other models, it made fewer overall errors and maintains high accuracy with consistent performance across both classes.

Discussion: The high effectiveness of the SVM proves that it is a good parameter with high-dimensional textual data. It produces marginal higher accuracy than Logistic Regression but the training and prediction are quite expensive in time. This comparison is an important point of difference because of this trade-off of a modest improvement in accuracy at the cost of a significant increment of computational cost.

3.5. Feedforward Neural Network (FFNN): A Deep Learning Perspective

3.5.1. Theoretical Framework

Artificial Neural Networks A Feedforward Neural Network (FFNN) is a family of Artificial Neural Networks in which the links between the nodes do not create a loop. It is composed of an input layer, one or more hidden layers and an output layer. The input entered flows in a one direction, input layer, followed by the hidden layers to an output layer. The neurons in a given layer link to the neurons in the adjacent layer and each of these connections is affixed a weight. Training helps the network discover such weights via a mechanism known as backpropagation wherein error between the prediction made by the network and the actual label value aids in updating the weights using a form of an optimization algorithm such as Adam. The activation functions (such as ReLU being used in hidden layers and Sigmoid in the binary output layer) add non-linearity enabling the network to learn complex patterns.

3.5.2. Implementation in Python

The implementation uses the Keras library, a high-level API for building and training neural networks. A simple sequential model is constructed with two hidden layers.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

print("\n--- Feed Forward Neural Network ---")

model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

y_train_numeric = y_train.apply(lambda x: 1 if x == 'REAL' else 0)
y_test_numeric = y_test.apply(lambda x: 1 if x == 'REAL' else 0)

start_time = time.time()
history = model.fit(X_train, y_train_numeric, epochs=10, batch_size=32, validation_split=0.2, verbose=1)
end_time = time.time()
print("Training Time:", end_time - start_time, "seconds")

loss, accuracy = model.evaluate(X_test, y_test_numeric, verbose=0)
print(f"Test Accuracy: {accuracy}")
```

```python
start_time = time.time()
y_pred_nn_proba = model.predict(X_test)
y_pred_nn = (y_pred_nn_proba > 0.5).astype("int").flatten()
end_time = time.time()
print("Prediction Time:", end_time - start_time, "seconds")

y_pred_nn_labels = ['REAL' if pred == 1 else 'FAKE' for pred in y_pred_nn]

print("Classification Report:\n", classification_report(y_test, y_pred_nn_labels))

cm_nn = confusion_matrix(y_test, y_pred_nn_labels)
sns.heatmap(cm_nn, annot=True, fmt='d', cmap='Blues', xticklabels=['Fake', 'Real'], yticklabels=['Fake', 'Real'])
plt.title("Confusion Matrix: Feed Forward Neural Network")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

3.5.3. Results and Evaluation

- **Accuracy:** 0.9202
- **Training Time:** 21.4366 seconds
- **Prediction Time:** 0.5723 seconds

Classification Report:

	precision	recall	f1-score	support
FAKE	0.93	0.91	0.92	633
REAL	0.91	0.93	0.92	634
accuracy			0.92	1267
macro avg	0.92	0.92	0.92	1267
weighted avg	0.92	0.92	0.92	1267

Confusion Matrix Analysis: The Feed Forward Neural Network demonstrates high overall accuracy with strong and balanced performance. It correctly identified 575 fake articles and 591 real articles, while misclassifying 58 fake articles as real and 43 real articles as fake. The performance is consistent across both classes, reflecting the model's ability to learn and generalize the underlying patterns effectively.

Discussion: FFNN demonstrates irresistible forces of deep learning, and performs at a highest level of accuracy. This however, is at a high computational cost. The training duration is huge compared to the rest of the models, which indicates the repetitive process of gradient descent during various epochs. It has superb predictive power, but the trade-off between a marginal improvement in accuracy and an enormous increase in the difficulty and time to train is one of the leitmotifs of the final analysis.

4. Comparative Benchmarking and Efficiency Analysis

This section summarizes both implications of the five models so as to carry out a complete comparison. It is aimed at stepping beyond an individual model performance to find the best algorithm using both predictive accuracy and computational efficiency of performances that serve as the dual criteria.

4.1. Synthesis of Performance Metrics

In order to make it easy to compare, the most important performance indicators of each of the five algorithms are presented in one table. This table gives a good overview of the trade-offs between the various approaches in the form of an at-a-glance visual representation.

Table 1: Model Performance Comparison

Model	Accuracy	F1-Score (FAKE)	Training Time (s)	Prediction Time (s)
Multinomial Naive Bayes	0.8768	0.88	0.17	0.02
K-Nearest Neighbors (KNN)	0.8721	0.87	0.02	4.33
Logistic Regression	0.9147	0.92	2.16	0.02
Support Vector Machine (SVM)	0.9281	0.93	54.14	9.76
Feedforward Neural Network	0.9202	0.92	21.43	0.57

4.2. Predictive Accuracy Showdown

The F1-scoring metric can also be a stronger assessment of predictive power than accuracy, particularly on the target output label of FAKE (precision = proportion of predicted fake that were actually fake; recall = proportion of actual fake that were correctly predicted), and indeed this measure often outperforms accuracy on this target label.

The results clearly show a tiered performance structure.

1. **Top Tier:** The highest results are the Support Vector Machine (SVM) and the Feedforward Neural Network (FFNN) of 0.93 and 0.92 F1-scores, respectively. Logistic Regression is also able to show such a high performance of F1-score of 0.92, which proves again the effectiveness of the linear model and neural architecture that are able to detect linear decision boundaries especially within the sparse and high-dimensional spaces that occur due to TF-IDF. The skill of SVM of finding out a maximum-margin hyperplane is very useful in those situations.
2. **Mid Tier:** The multinomial Naive Bayes is one of the good performers since it has an F1-score of 0.88. Its simplistic assumption that features are independent items (not completely true in natural language), it remains a sound and computationally - efficient benchmark of the text classification activities.
3. **Lower Tier:** K-Nearest Neighbors (KNN) lags behind with F1-score equals 0.87. Its application would probably be complicated by the data dimensionality, which blurs the notion of distance, and interferes with its capability of finding really significant neighbors--a so-called curse of dimensionality.

4.3. Computational Cost Analysis

Computational cost analysis shows dramatic differences among the algorithms especially in training time.

- **Instantaneous Training:** According to KNN, training takes insignificant time since it is a lazy learner; it kicks its can to the prediction time.
- **Extremely Fast Training:** The Multinomial Naive Bayes model is the quickest among the models to be trained and takes less than 0.1 s. This is because of its easy probabilistic calculation.
- **Fast Training:** Logistic Regression is one other option that is fast, in less than 4 seconds. It makes an optimized solution.
- **Moderate Training:** The linear SVM is comparatively slow; it requires more than 12 seconds to converge and find the optimal hyperplane.
- **Slow Training:** The slowest training is the Feedforward Neural Network which takes close to 2 minutes to train 10 epochs. This is because the backpropagation is expensive and updating of weights has to be done within the numerous parameters of the network.

Differences involving prediction times are also highly varying. Most of the models are almost instant, but KNN is a significant exception. This is because it takes more than 40 seconds to make a prediction and so, it cannot be used in any application that needs real-time or even near-real-time classification. The remaining models (including the intricate FFNN) are able to forecast the whole testing set in less than a second (the SVM is a bit slower, yet it is still exceptionally fast).

This is well depicted by the results in a performance frontier in terms of the accuracy with a huge computational cost. The linear models, namely Logistic Regression and SVM, take a sweet spot in that their performance is almost as good as the far more expensive neural network, yet its cost to train is a fraction of that of the latter. The transition between Logistic Regression and the FFNN delivers a difference of only 1% higher in F1-score but a difference of ~30x longer at training it. This shows one of the (because the analysis involves only one experimental condition) laws of diminishing returns: in this particular problem formulation, the effect of

increased complexity in the neural network does not come with an equivalent increase in predictive power.

5. Conclusion and Strategic Recommendations

5.1. The Verdict on the Most Efficient Algorithm

Focusing on a thorough consideration of predictive efficiency and computational cost, **Logistic Regression** is singled out in terms of being the most efficient algorithm in terms of the overall efficiency of the proposed fake news detection task.

Although Feedforward Neural Network demonstrated the best precision, the minimal difference, being less than 1 percent, cannot compensate for the far greater training duration and morass of building it in comparison to the Logistic Regression. Equally, the Support Vector Machine, although another high performer, takes more time to train and predict than Logistic Regression at the expense of insignificant increase in accuracy. Multinomial Naive Bayes is the quickest one, but it does not predict well, compared to the linear models. K-Nearest Neighbors is evidently time efficient since its time of prediction is quite slow, and it is less accurate.

Thus, Logistic Regression as the best tradeoff can be used. It achieves elite predictive precision (95% F1-score) at minimal computational cost, and thus it is an effective, efficient and workable solution to this text classification.

5.2. Recommendations for Future Work

This project serves as a robust foundation, but several avenues exist for further exploration and improvement.

- **Cross-Dataset Validation:** Violations are done on a single dataset in the current model. To achieve a good generalization it may be considered to test the models on other publicly available fake news datasets to test their effectiveness on unseen data that will not be overfitted to the appearance and content of the present one.
- **Real-Time Fake News Detection System:** The useful extension may be turning the existing offline pipeline into a real-time system. This would entail combining a front-end interface, and back-end model predictions so that people would feed news snippets and be given feedback on the spot. The optimization on fast inference of models would also be needed.
- **Advanced Architectures:** The next step can be treating more complex neural network architecture specific to text like Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs) and Transformer-based models like BERT. These models are more adept at modeling word and context dependency in the sentence and this is quite likely to enhance performance of classification compared to the existing feed-forwards network.

References

1. Detecting Fake News Dataset - Kaggle, https://www.kaggle.com/datasets/amirmotefaker/detecting-fake-news-dataset/data
2. 100 Days of Machine Learning - CampusX, https://youtube.com/playlist?list=PLKnIA16_Rmvbr7zKYQuBfsVkjoLcJgxHH&si=8VW4tqzcYHlrk6rj
3. Text Classification using Logistic Regression - GeeksforGeeks, https://www.geeksforgeeks.org/machine-learning/text-classification-using-logistic-regression/

4. Getting started with Text Preprocessing - Kaggle, https://www.kaggle.com/code/sudalairajkumar/getting-started-with-text-preprocessing
5. Stemming vs. Lemmatization in NLP | Towards Data Science, https://towardsdatascience.com/stemming-vs-lemmatization-in-nlp-dea008600a0/
6. Bag-of-words vs TF-IDF - GeeksforGeeks, https://www.geeksforgeeks.org/nlp/bag-of-words-vs-tf-idf/
7. Naive Bayes Classifier Tutorial: with Python Scikit-learn - DataCamp, https://www.datacamp.com/tutorial/naive-bayes-scikit-learn
8. K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks, https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/
9. KNN Classifier in Python: Implementation, Features and Application - Analytics Vidhya, https://www.analyticsvidhya.com/blog/2021/01/a-quick-introduction-to-k-nearest-neighbor-knn-classification-using-python/
10. Implementing SVM and Kernel SVM with Python's Scikit-Learn - GeeksforGeeks, https://www.geeksforgeeks.org/machine-learning/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/
11. Practical Text Classification With Python and Keras, https://realpython.com/python-keras-text-classification/
12. FeedForward Neural Network using TensorFlow, Keras - YouTube, https://www.youtube.com/watch?v=Lof9jieVF0s