Loyola Marymount University

VERIFACT

By : Hardi Dave, Dhairya Patel, and Avi Patel

INTRODUCTION

- **This capstone project focuses on leveraging machine learning techniques to analyze the authenticity of news shared on social media platforms.**
- **Using Kaggle dataset to train and test the models identifying Positive & Negative News.**
- **This project addresses the growing spread of fake news in the digital age by tackling societal challenges of confusion, mistrust, and harm caused by false information.**

PROBLEM

- **Persona: Mark, 28, Social Media Influencer**
- **Quote: "As a social media influencer with a large following, I can't afford to accidentally share misinformation. I need a reliable way to quickly verify the authenticity of news articles and content before posting, so I can maintain trust and credibility with my audience."**
- **Key Points:**
    - **Time-consuming manual fact-checking**
    - **Risk of damaging credibility and trust with his large social media followers**
    - **Overwhelming volume of information to verify**

SOLUTION

- **For Mark:**
  - **AI-powered news and content authenticity checker**
  - **Quick analysis of articles and social media posts**
  - **Detailed reports with credibility scores and explanations**

UNIQUE VALUE PROPOSITION

- **Key differentiators:**
  - **User-friendly interface for quick checks**
  - **Detailed analysis reports for in-depth understanding**
  - **Continuous learning from user feedback**
  - **Collaborative flagging system**

ADDITIONAL PERSONAS

Anna, 45, Concerned Parent

- **Wants to ensure his children access reliable information online**

Emma, 32, Social Media Manager

- **Needs to verify content before sharing on company platforms**

Prof. Johnson, 55, University Lecturer

- **Encourages students to fact-check their sources**

MVP

- **AI-powered authenticity analysis**
- **Basic explanation of the analysis result**
- **User registration and login**
- **Text input for news articles and social media posts**
- **Credibility score display**

Technologies

- **Tools: Jupyter Notebook**
- **Technology: Python language**
- **Data Pre-Processing: Pandas, Numpy, NLTK, Doc2Vec**
- **ML Models: Naive Byes, SVM, Decision Tree, KNN, Logistic regression, Ensemble models, TensorFlow/Keras, Neural Network**

USER STORIES

"As a concerned citizen, I want to quickly verify the authenticity of the news articles I come across, so that I can avoid sharing misinformation with my friends and family"

"As a journalist, I want to see a detailed breakdown of why content was classified as fake or real."

"As a student, I want to verify the information I use in my research papers, so that I can confidently cite reliable sources and improve the quality of my academic work."

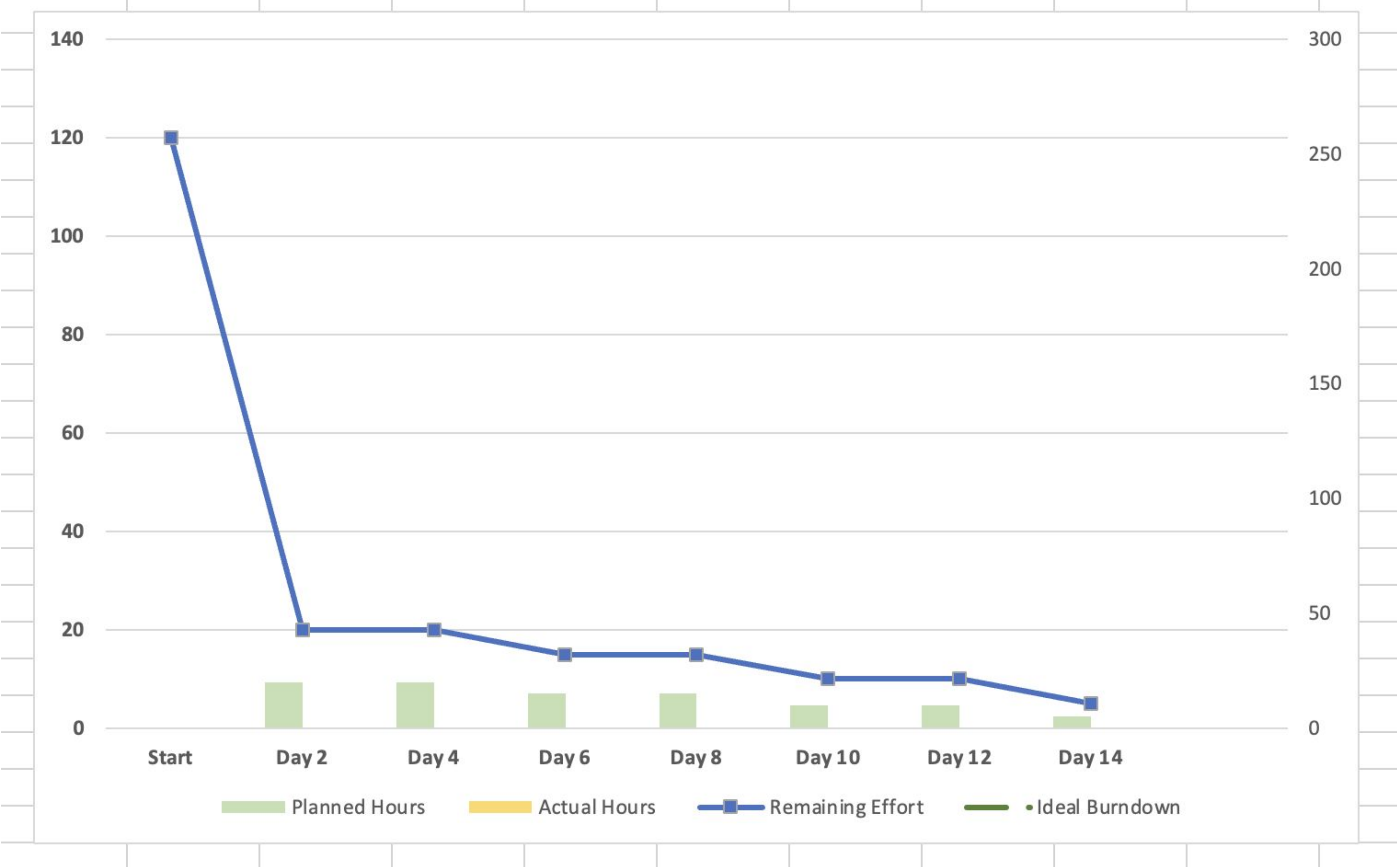
PROJECT BACKLOG

	Features/Task	Story Point	Status
1	Data preparation and model training	8	Completed
2	Basic web interface	5	In Progress
3	URL input support	6	In Progress
4	Analysis report	4	Pending

TEST CASES

	Test	Category	Pass Criteria	Status
1	Test with various text lengths	Input Validation	System should handle all text lengths without errors	Completed
2	Test with special characters and different languages	Input Validation	System should correctly process and analyze input regardless of language or characters	Completed
3	Test user registration with valid inputs	Authentication	Registration should succeed with valid inputs	In Progress
4	Test user registration with invalid inputs	Authentication	Registration should fail with invalid inputs	In Progress
5	Test with known fake news dataset	Analysis Accuracy	System should correctly identify fake news	Completed
6	Test with known real news dataset	Analysis Accuracy	System should correctly identify real news	Completed
7	Test response time for analysis	Performance	System should analyze input and return results within an acceptable time frame	Pending
8	Test concurrent user handling	Performance	System should handle multiple users simultaneously without performance degradation	Pending

SPRINT BURNDOWN CHART



RETROSPECTIVE

What Went Well:

- **Good Teamwork, Taking iterative approach allowed for flexibility and adaptation to changing project needs.**
- **Successfully deployed machine learning models into production.**

Areas for Improvement:

- **Enhance prediction accuracy through feature engineering.**
- **Optimize processing speed for real-time analysis**

Lessons Learned:

- **Regular sync-ups and status updates drive project momentum.**
- **Early integration testing helps identify and resolve issues faster**

RESULTS

Result	Column1	Column2	Column3
Model	**Accuracy**	**Cohen's Kappa**	**Matthews Corr**
SVM	0.9885	0.971776	0.971778
Logistic Regression	0.9884	0.976	0.976
Neural Network	0.9886	N/A	N/A
Voting Classifer (Ensemble)	0.8702	N/A	N/A
Naïve Byes	0.9709	0.941	0.942
KNN	0.8661	0.73	0.7352
Decision Tree	0.75905	0.516	0.5163

SCREENSHOTS OF OUTPUT

Data cleaning

1.1. Removing HTML tags or unwanted characters 1.1.1. Eliminating special characters, punctuation 1.1.2. remove common English stopwords. 1.2. Converting text to lowercase 1.3. finding rows with missing values 1.3.1. Filling missing values with a plac
Removing rows with unnecessary missing values 1.4. Text Length (Characters) 1.4.1. Word Count:

```
# Function to clean text
def clean_text(text):
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # Remove punctuation and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Convert text to lowercase
    text = text.lower().strip()
    return text

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

[22]

```
# Download the stopwords dataset if not already downloaded
nltk.download('stopwords')
# Download the Punkt sentence tokenizer model
nltk.download('punkt')

# Load stop words
stop_words = set(stopwords.words('english'))

# Function to remove stop words
def remove_stop_words(text):
    word_tokens = word_tokenize(text)
    filtered_text = [word for word in word_tokens if word not in stop_words]
    return ' '.join(filtered_text)


df['cleaned_title_no_stopwords'] = df['cleaned_title'].apply(remove_stop_words)
df['cleaned_text_no_stopwords'] = df['cleaned_text'].apply(remove_stop_words)
```

[23]

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
df.drop(['cleaned_text', 'cleaned_title',], axis=1, inplace=True)
df.head()
```

[24]

	title	text	subject	date	label	cleaned_title_no_stopwords	cleaned_text_no_stopwords
0	OUTRAGE! HOW REFUGEE RESETTLEMENT Is Using Uni...	Admitting Somalis who d been settled for year...	politics	Dec 9, 2016	1	outrage refugee resettlement using united stat...	admitting somalis settled years pakistan like ...
1	ARMY THREATENS GREEN BERET WAR HERO WITH COURT...	The Army can t be bothered with defending or p...	left-news	Jun 12, 2015	1	army threatens green beret war hero court mart...	army bothered defending protecting war heroes ...
2	Bill Maher Hilariously Hammers 'Evil' Ted Cru...	Jimmy Kimmel couldn t stop laughing as Bill Ma...	News	February 10, 2016	1	bill maher hilariously hammers evil ted cruz j...	jimmy kimmel stop laughing bill maher repeated...
3	BREAKING: PRESIDENT-ELECT TRUMP Meets With Pre...	Obama: Time for us to come together, work toge...	politics	Nov 10, 2016	1	breaking presidentelect trump meets president ...	obama time us come together work together deal...
4	As Catalan vote looms, jailed leader offers ol...	MADRID/BARCELONA (Reuters) - The jailed leader...	worldnews	December 18, 2017	0	catalan vote looms jailed leader offers olive ...	madridbarcelona reuters jailed leader cataloni...

Steemming

```
#stemming
# Initialize the Porter Stemmer
stemmer = PorterStemmer()

#stemming to a text
def stem_text(text):
    word_tokens = word_tokenize(text)
    stemmed_text = [stemmer.stem(word) for word in word_tokens]
    return ' '.join(stemmed_text)

#progress visualization
def stem_with_progress(data, column_name):
    stemmed_data = []
    total = len(data)
    print("Starting stemming process...")
    for i, text in enumerate(data[column_name], 1):
        stemmed_data.append(stem_text(text))
        if i % 100 == 0 or i == total:
            sys.stdout.write('\rProgress: {0:.2f}%'.format(100 * i/total))
            sys.stdout.flush()
    print("\nStemming process completed.")
    return stemmed_data


df['stemmed_title'] = stem_with_progress(df, 'cleaned_title_no_stopwords')
df['stemmed_text'] = stem_with_progress(df, 'cleaned_text_no_stopwords')
```

[25]

```
Starting stemming process...
Progress: 100.00%
Stemming process completed.
Starting stemming process...
Progress: 100.00%
Stemming process completed.
```

Lemmetization

```
#lemmetization
# Initialize the WordNet Lemmatizer
# Download the resource

nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()

# Function to convert NLTK's part-of-speech tags to WordNet's part-of-speech tags
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # Default to noun if unknown

# apply lemmatization to a text
def lemmatize_text(text):
    word_tokens = word_tokenize(text)
    pos_tagged_tokens = pos_tag(word_tokens)
    lemmatized_text = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tagged_tokens]
    return ' '.join(lemmatized_text)

# progress visualization
def lemmatize_with_progress(data, column_name):
    lemmatized_data = []
    total = len(data)
    print("Starting lemmatization process...")
    for i, text in enumerate(data[column_name], 1):
        lemmatized_data.append(lemmatize_text(text))
        if i % 100 == 0 or i == total:  # Update progress every 100 items or at the end
            sys.stdout.write('\rProgress: {0:.2f}%'.format(100 * i/total))
            sys.stdout.flush()
    print("\nLemmatization process \completed.")
    return lemmatized_data


df['lemmatized_title'] = lemmatize_with_progress(df, 'stemmed_title')
df['lemmatized_text'] = lemmatize_with_progress(df, 'stemmed_text')
```

[27]

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
Starting lemmatization process...
Progress: 100.00%
Lemmatization process \completed.
Starting lemmatization process...
Progress: 100.00%
Lemmatization process \completed.
```

Sentiment Analysis

```
from textblob import TextBlob
df['sentiment'] = df['lemmatized_text'].apply(lambda text: TextBlob(text).sentiment.polarity)
```

[36]

```
for index, row in df.iterrows():
    sentiment_score = row['sentiment']
    sentiment_score2 = row['label']
    print(f"Document {index} has a sentiment polarity of {sentiment_score} and the sentiment score is {sentiment_score2}")

    if sentiment_score < 0:
        print("The sentiment is negative.")
    elif sentiment_score > 0:
        print("The sentiment is positive.")
    else:
        print("The sentiment is neutral.")
```

[37]

```
Streaming output truncated to the last 5000 lines.
Document 42398 has a sentiment polarity of 0.29642857142857143 and the sentiment score is 0
The sentiment is positive.
Document 42399 has a sentiment polarity of 0.0024891774891774793 and the sentiment score is 1
The sentiment is positive.
Document 42400 has a sentiment polarity of 0.13190476190476189 and the sentiment score is 1
The sentiment is positive.
Document 42401 has a sentiment polarity of 0.020000000000000007 and the sentiment score is 1
The sentiment is positive.
Document 42402 has a sentiment polarity of 0.2767232767232767 and the sentiment score is 1
The sentiment is positive.
Document 42403 has a sentiment polarity of 0.03398268398268399 and the sentiment score is 1
The sentiment is positive.
Document 42404 has a sentiment polarity of 0.06688311688311688 and the sentiment score is 1
The sentiment is positive.
Document 42405 has a sentiment polarity of -0.09722222222222222 and the sentiment score is 1
The sentiment is negative.
Document 42406 has a sentiment polarity of 0.050396825396825405 and the sentiment score is 0
The sentiment is positive.
Document 42407 has a sentiment polarity of -0.041666666666666667 and the sentiment score is 1
The sentiment is negative.
Document 42408 has a sentiment polarity of 0.03636363636363637 and the sentiment score is 0
The sentiment is positive.
Document 42409 has a sentiment polarity of 0.4394736842105263 and the sentiment score is 1
The sentiment is positive.
...
Document 44896 has a sentiment polarity of -0.07500000000000001 and the sentiment score is 0
The sentiment is negative.
Document 44897 has a sentiment polarity of 0.06496392496392496 and the sentiment score is 1
The sentiment is positive.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Vectorization

```
from gensim.models.doc2vec import TaggedDocument
from gensim.models import Doc2Vec

documents = [TaggedDocument(words=text.split(), tags=[i]) for i, text in enumerate(df['lemmatized_title'])]
```

[40]

```
lemmatized_docs = (df['subject']+" " +df['lemmatized_title'] + " " + df['lemmatized_text']+" "+ df['sentiment']).apply(lambda x: x.split())
print(lemmatized_docs)

tagged_title_data = [TaggedDocument(words=words, tags=[str(i)]) for i, words in enumerate(lemmatized_docs)]

max_epochs = 100
vec_size = 300
alpha = 0.025

model = Doc2Vec(vector_size=vec_size,
                alpha=alpha,
                min_alpha=0.00025,  # Gradual decay to the minimum alpha
                min_count=5,
                window=10,
                dm=1,
                )

model.build_vocab(tagged_title_data)
print('Training Doc2Vec Model')

for epoch in range(max_epochs):
    print(f'Training epoch {epoch + 1}/{max_epochs}')
    model.train(tagged_title_data,
                total_examples=model.corpus_count,
                epochs=1)  # Train for one epoch at a time
    model.alpha -= (alpha - model.min_alpha) / max_epochs  # Decrease the learning rate
    model.min_alpha = model.alpha  # Fix the learning rate, no decay

# Save the model
model.save("d2v_all_all21.model")
print("Model Saved")
```

[41]

```
0        [politics, outrag, refuge, resettl, use, unit,...
1        [left-news, armi, threaten, green, beret, war,...
2        [News, bill, maher, hilari, hammer, evil, ted,...
3        [politics, break, presidentelect, trump, meet,...
4        [worldnews, catalan, vote, loom, jail, leader,...
                               ...
44893    [politicsNews, exdemocrat, leader, mull, drop,...
44894    [News, hillari, gloriou, frozen, pun, total, m...
44895    [worldnews, argentina, macri, vow, pursu, tax,...
44896    [worldnews, irma, head, westnorthwest, pas, ca...
44897    [News, nra, get, blast, livetweet, obama, town...
Length: 44888, dtype: object
Training Doc2Vec Model
Training epoch 1/100
WARNING:gensim.models.word2vec:Effective 'alpha' higher than previous training cycles
Training epoch 2/100
Training epoch 3/100
```

Naive bayes

```python
# naive bayes using sckit learn
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import accuracy_score, classification_report, matthews_corrcoef

labels = df['label'].values


np_combined_vectors += np.abs(np_combined_vectors.min())

X_train, X_test, y_train, y_test = train_test_split(np_combined_vectors, labels, test_size=0.2, random_state=42)

nb_classifier = MultinomialNB()

nb_classifier.fit(X_train, y_train)

y_pred = nb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

kappa = cohen_kappa_score(y_test, y_pred)
print(f"Cohen's Kappa Score: {kappa}")

mcc = matthews_corrcoef(y_test, y_pred)
print(f"Matthews Correlation Coefficient (MCC): {mcc}")
```

[47]

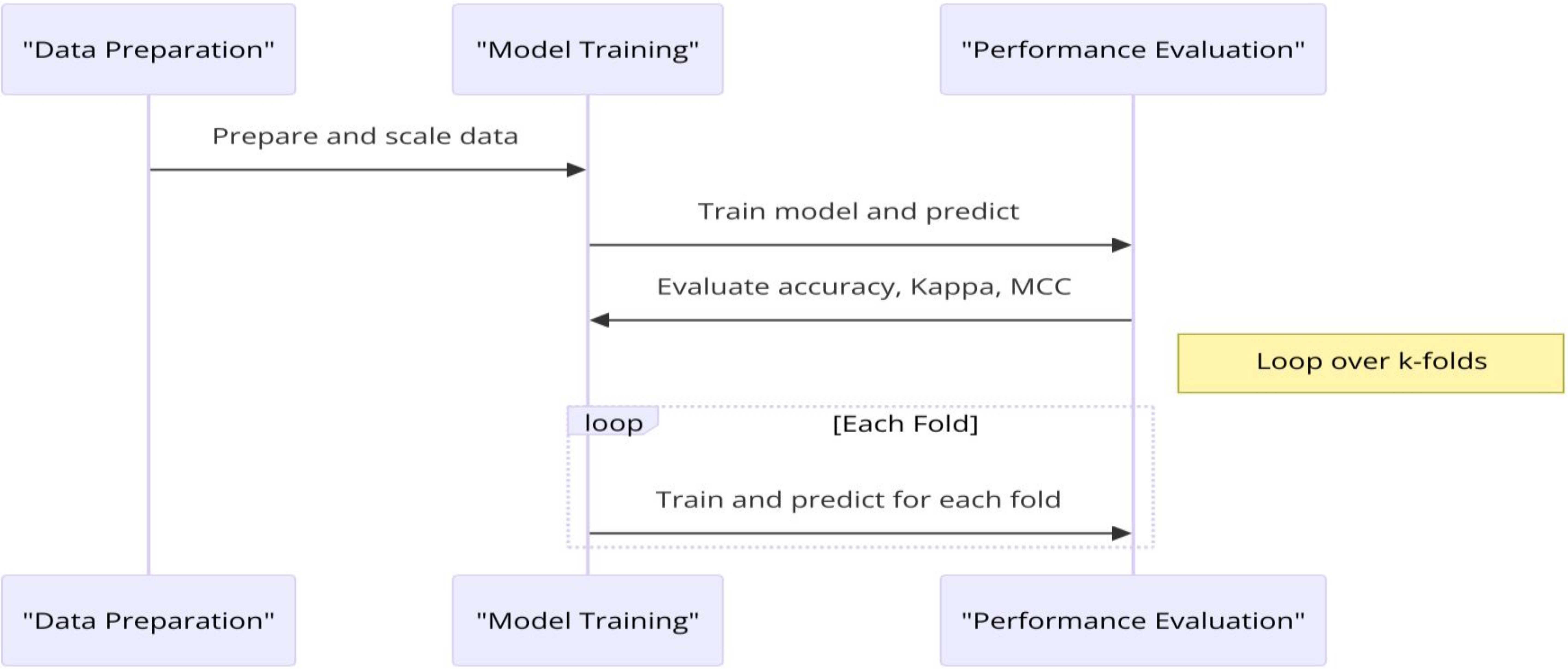
```
Test Accuracy: 0.9709289374025395
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      4293
           1       0.98      0.96      0.97      4685

    accuracy                           0.97      8978
   macro avg       0.97      0.97      0.97      8978
weighted avg       0.97      0.97      0.97      8978

Cohen's Kappa Score: 0.9417995678590754
Matthews Correlation Coefficient (MCC): 0.9420021522136142
```

Support Vector Machine (SVM)

```
#SVM using ski-it learn
from sklearn.svm import SVC

labels = df['label'].values

X_train, X_test, y_train, y_test = train_test_split(np_combined_vectors, labels, test_size=0.2, random_state=42)

svm_classifier = SVC(kernel='linear', max_iter=10000, tol=1e-3)


svm_classifier.fit(X_train, y_train)

y_pred = svm_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))


mcc = matthews_corrcoef(y_test, y_pred)
print(f"Matthews Correlation Coefficient: {mcc}")

kappa = cohen_kappa_score(y_test, y_pred)
print(f"Cohen's Kappa Score: {kappa}")
```

[48]

```
Test Accuracy: 0.9778347070617064
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      4293
           1       0.98      0.98      0.98      4685

    accuracy                           0.98      8978
   macro avg       0.98      0.98      0.98      8978
weighted avg       0.98      0.98      0.98      8978

Matthews Correlation Coefficient: 0.9555842625768918
Cohen's Kappa Score: 0.9555773817512858
```


"Data Preparation"
"Model Training"
"Performance Evaluation"
Prepare and scale data
Train model and predict
Evaluate accuracy, Kappa, MCC
Loop over k-folds
loop
[Each Fold]
Train and predict for each fold
"Data Preparation"
"Model Training"
"Performance Evaluation"

Decision Tree classifier

```
# decision tree using kfold from skilearn
from sklearn.tree import DecisionTreeClassifier

scaler = StandardScaler()
np_combined_vectors_scaled = scaler.fit_transform(np_combined_vectors)

k_folds = 5
kf = KFold(n_splits=k_folds, shuffle=True, random_state=2)

accuracies = []
kappa_scores = []
mcc_scores = []

for fold, (train_index, test_index) in enumerate(kf.split(np_combined_vectors_scaled), 1):
    X_train, X_test = np_combined_vectors_scaled[train_index], np_combined_vectors_scaled[test_index]
    y_train, y_test = labels[train_index], labels[test_index]

    dt_classifier = DecisionTreeClassifier(random_state=2)

    dt_classifier.fit(X_train, y_train)

    y_pred = dt_classifier.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    kappa = cohen_kappa_score(y_test, y_pred)
    mcc = matthews_corrcoef(y_test, y_pred)

    accuracies.append(accuracy)
    kappa_scores.append(kappa)
    mcc_scores.append(mcc)

    print(f"Fold {fold}")
    print(f"Accuracy: {accuracy}")
    print(f"Cohen's Kappa Score: {kappa}")
    print(f"Matthews Correlation Coefficient: {mcc}")
    print("-" * 30)

print(f"Average Accuracy across all folds: {np.mean(accuracies)}")
print(f"Average Cohen's Kappa Score across all folds: {np.mean(kappa_scores)}")
print(f"Average Matthews Correlation Coefficient across all folds: {np.mean(mcc_scores)}")
```

[50]

```
Fold 1
Accuracy: 0.7532858097571842
Cohen's Kappa Score: 0.5048762528619363
Matthews Correlation Coefficient: 0.5051416604620014
------------------------------
Fold 2
Accuracy: 0.7566273112051682
Cohen's Kappa Score: 0.5116685422159907
Matthews Correlation Coefficient: 0.5120083966785136
------------------------------
Fold 3
Accuracy: 0.7554020940075741
Cohen's Kappa Score: 0.5088406347983995
Matthews Correlation Coefficient: 0.509538958856758
------------------------------
Fold 4
Accuracy: 0.7576027626155731
Cohen's Kappa Score: 0.5128468443475755
Matthews Correlation Coefficient: 0.5134145591418696
------------------------------
Fold 5
Accuracy: 0.7590509078756823
Cohen's Kappa Score: 0.516073313558981
Matthews Correlation Coefficient: 0.51636030054038
------------------------------
Average Accuracy across all folds: 0.7563937770922364
Average Cohen's Kappa Score across all folds: 0.5108611175565766
Average Matthews Correlation Coefficient across all folds: 0.5112927751359045
```

K-Nearest-Neighbour (KNN)

```python
#knn with folds
from sklearn.neighbors import KNeighborsClassifier

scaler = StandardScaler()
np_combined_vectors_scaled = scaler.fit_transform(np_combined_vectors)


k_folds = 5
kf = KFold(n_splits=k_folds, shuffle=True, random_state=42)


fold_accuracies = []
fold_kappa_scores = []
fold_mcc_scores = []

for fold, (train_index, test_index) in enumerate(kf.split(np_combined_vectors_scaled), 1):

    X_train, X_test = np_combined_vectors_scaled[train_index], np_combined_vectors_scaled[test_index]
    y_train, y_test = labels[train_index], labels[test_index]

    knn_classifier = KNeighborsClassifier(n_neighbors=2)

    knn_classifier.fit(X_train, y_train)

    y_pred = knn_classifier.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    kappa = cohen_kappa_score(y_test, y_pred)
    mcc = matthews_corrcoef(y_test, y_pred)


    fold_accuracies.append(accuracy)
    fold_kappa_scores.append(kappa)
    fold_mcc_scores.append(mcc)


    print(f"Fold {fold}")
    print(f"Accuracy: {accuracy}")
    print(f"Cohen's Kappa Score: {kappa}")
    print(f"Matthews Correlation Coefficient: {mcc}")
    print("-" * 30)

print(f"Average Accuracy across all folds: {np.mean(fold_accuracies)}")
print(f"Average Cohen's Kappa Score across all folds: {np.mean(fold_kappa_scores)}")
print(f"Average Matthews Correlation Coefficient across all folds: {np.mean(fold_mcc_scores)}")
```

[51]

```
Fold 1
Accuracy: 0.8661171753174426
Cohen's Kappa Score: 0.7303776322554374
Matthews Correlation Coefficient: 0.7352426308943919
------------------------------
Fold 2
Accuracy: 0.8622187569614613
Cohen's Kappa Score: 0.7221923741562988
Matthews Correlation Coefficient: 0.7277042106790089
------------------------------
Fold 3
Accuracy: 0.8592114056582758
Cohen's Kappa Score: 0.7170948878804284
Matthews Correlation Coefficient: 0.7247099677770744
------------------------------
Fold 4
Accuracy: 0.8620920129219115
Cohen's Kappa Score: 0.72039217010341
Matthews Correlation Coefficient: 0.7261821378322965
------------------------------
Fold 5
Accuracy: 0.8500612676840815
Cohen's Kappa Score: 0.6979072291651343
Matthews Correlation Coefficient: 0.7051740035099472
```

Logistic regression

```
from sklearn.linear_model import LogisticRegression

k_folds = 5
kf = KFold(n_splits=k_folds, shuffle=True, random_state=42)


fold_accuracies = []
fold_kappa_scores = []
fold_mcc_scores = []


for train_index, test_index in kf.split(np_combined_vectors):
    X_train, X_test = np_combined_vectors[train_index], np_combined_vectors[test_index]
    y_train, y_test = labels[train_index], labels[test_index]

    logistic_model = LogisticRegression(max_iter=1000, random_state=42)

    logistic_model.fit(X_train, y_train)

    y_pred = logistic_model.predict(X_test)

    fold_accuracies.append(accuracy_score(y_test, y_pred))
    fold_kappa_scores.append(cohen_kappa_score(y_test, y_pred))
    fold_mcc_scores.append(matthews_corrcoef(y_test, y_pred))

print(f"Average Logistic Regression Test Accuracy: {np.mean(fold_accuracies)}")
print(f"Average Cohen's Kappa Score across all folds: {np.mean(fold_kappa_scores)}")
print(f"Average Matthews Correlation Coefficient across all folds: {np.mean(fold_mcc_scores)}")
```

[54]

```
Average Logistic Regression Test Accuracy: 0.9884379856117537
Average Cohen's Kappa Score across all folds: 0.9768255923343355
Average Matthews Correlation Coefficient across all folds: 0.9768289101123674
```

Ensemble Model

```python
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np



X_train, X_test, y_train, y_test = train_test_split(np_combined_vectors, labels, test_size=0.2, random_state=42)

log_clf = LogisticRegression(random_state=42)
tree_clf = DecisionTreeClassifier(random_state=42)
knn_clf = KNeighborsClassifier()

ensemble_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('dt', tree_clf), ('knn', knn_clf)],
    voting='soft'  # or 'soft' if you want to weigh probabilities for classification tasks
)

ensemble_clf.fit(X_train, y_train)


y_pred = ensemble_clf.predict(X_test)
print("Ensemble Model Accuracy:", accuracy_score(y_test, y_pred))
```

[57]

```
... Ensemble Model Accuracy: 0.8702383604366228
```

Neural Network

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
np.random.seed(42)
tf.random.set_seed(42)


X_train, X_test, y_train, y_test = train_test_split(np_combined_vectors, labels, test_size=0.30, random_state=42)



model = Sequential([
    Dense(256, input_shape=(X_train.shape[1],)),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.5),
    Dense(128),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.5),
    Dense(64),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.4),
    Dense(32),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])


model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=7, restore_best_weights=True)

history = model.fit(X_train, y_train, epochs=40, batch_size=32,
                    validation_data=(X_test, y_test),
                    callbacks=[early_stopping],
                    verbose=1)


test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f"\nTest Accuracy: {test_acc}")

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Over Epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
```

[58]

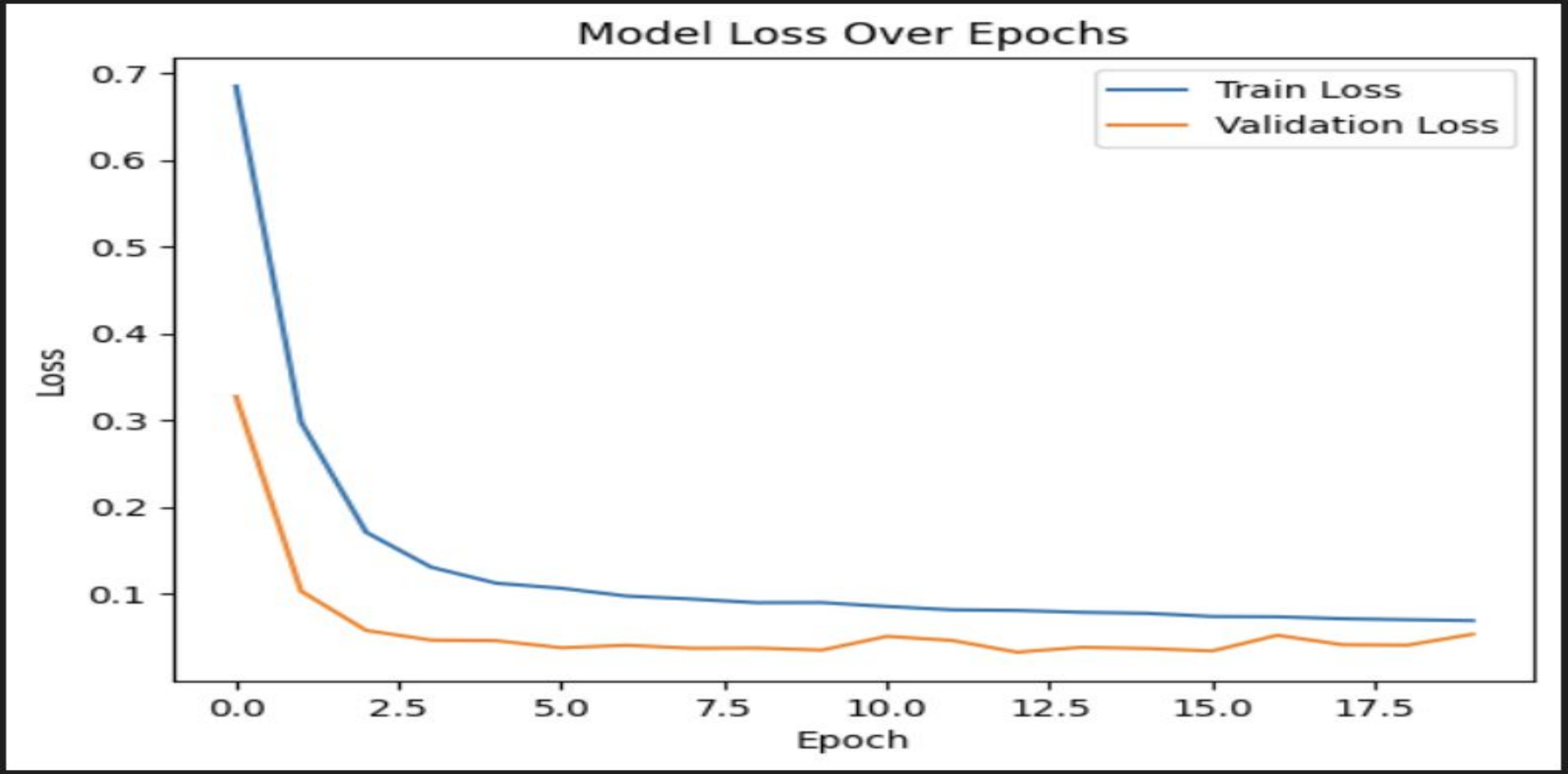
```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	153,856
leaky_re_lu (LeakyReLU)	(None, 256)	0
batch_normalization (BatchNormalization)	(None, 256)	1,024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
leaky_re_lu_1 (LeakyReLU)	(None, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
leaky_re_lu_2 (LeakyReLU)	(None, 64)	0
batch_normalization_2 (BatchNormalization)	(None, 64)	256
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 32)	2,080
leaky_re_lu_3 (LeakyReLU)	(None, 32)	0
batch_normalization_3 (BatchNormalization)	(None, 32)	128
dropout_3 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 1)	33

```
Total params: 199,041 (777.50 KB)

Trainable params: 198,081 (773.75 KB)

Non-trainable params: 960 (3.75 KB)

Epoch 1/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.5729 - loss: 0.8126 - val_accuracy: 0.8477 - val_loss: 0.3274
Epoch 2/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 10s 10ms/step - accuracy: 0.8396 - loss: 0.3563 - val_accuracy: 0.9730 - val_loss: 0.1029
Epoch 3/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 13ms/step - accuracy: 0.9288 - loss: 0.1884 - val_accuracy: 0.9839 - val_loss: 0.0577
Epoch 4/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.9518 - loss: 0.1359 - val_accuracy: 0.9869 - val_loss: 0.0465
Epoch 5/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 13ms/step - accuracy: 0.9581 - loss: 0.1149 - val_accuracy: 0.9844 - val_loss: 0.0459
Epoch 6/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.9598 - loss: 0.1082 - val_accuracy: 0.9880 - val_loss: 0.0377
Epoch 7/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - accuracy: 0.9656 - loss: 0.0959 - val_accuracy: 0.9867 - val_loss: 0.0406
Epoch 8/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.9674 - loss: 0.0946 - val_accuracy: 0.9882 - val_loss: 0.0372
Epoch 9/40
```

```
982/982 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.5729 - loss: 0.8126 - val_accuracy: 0.8477 - val_loss: 0.3274
Epoch 2/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 10s 10ms/step - accuracy: 0.8396 - loss: 0.3563 - val_accuracy: 0.9730 - val_loss: 0.1029
Epoch 3/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 13ms/step - accuracy: 0.9288 - loss: 0.1884 - val_accuracy: 0.9839 - val_loss: 0.0577
Epoch 4/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.9518 - loss: 0.1359 - val_accuracy: 0.9869 - val_loss: 0.0465
Epoch 5/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 13ms/step - accuracy: 0.9581 - loss: 0.1149 - val_accuracy: 0.9844 - val_loss: 0.0459
Epoch 6/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.9598 - loss: 0.1082 - val_accuracy: 0.9880 - val_loss: 0.0377
Epoch 7/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - accuracy: 0.9656 - loss: 0.0959 - val_accuracy: 0.9867 - val_loss: 0.0406
Epoch 8/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.9674 - loss: 0.0946 - val_accuracy: 0.9882 - val_loss: 0.0372
Epoch 9/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - accuracy: 0.9686 - loss: 0.0890 - val_accuracy: 0.9877 - val_loss: 0.0375
Epoch 10/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.9671 - loss: 0.0906 - val_accuracy: 0.9885 - val_loss: 0.0350
Epoch 11/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - accuracy: 0.9712 - loss: 0.0852 - val_accuracy: 0.9828 - val_loss: 0.0509
Epoch 12/40
982/982 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - accuracy: 0.9715 - loss: 0.0814 - val_accuracy: 0.9854 - val_loss: 0.0462
Epoch 13/40
...
982/982 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - accuracy: 0.9764 - loss: 0.0683 - val_accuracy: 0.9823 - val_loss: 0.0533
421/421 - 1s - 2ms/step - accuracy: 0.9886 - loss: 0.0327

Test Accuracy: 0.9886388778686523
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...




```
Enhanced model saved successfully.
```


Thank you