

APRIL 4, 2024

ZAINA FIROSH KAMAL – 100943331

PREYASKUMAR JAYSWAL – 100946767

DECLAN TREVOR KINTU – 100944330

ASSIGNMENT 2

AIDI 2000

GROUP ASSIGNMENT
PROFESSOR SAJEEVA SALGADOE



Table of Contents

INTRODUCTION:	2
MACHINE LEARNING MODELS:.....	2
<i>Model 1: Transfer Learning.</i>	2
<i>Model 2: Word Embedding.</i>	5
<i>Model 3: Logistic Regression.</i>	8
<i>Model 4: Ensemble Learning.</i>	11
OUR RECOMMENDATION:.....	14

INTRODUCTION:

For this assignment, we were tasked with creating four different Machine Learning models to train on a sentiment analysis dataset and evaluate their performances. Our group chose to perform the following four ML algorithms:

- 1) Transfer Learning.
- 2) Word Embedding.
- 3) Logistic Regression.
- 4) An Ensemble Model that combines results from Random Forest classification and Gradient-Boosted classification.

MACHINE LEARNING MODELS:

This is how we executed each model.

Model 1: Transfer Learning.

Load necessary libraries.

```
# Generic
import pandas as pd
from tensorflow.keras.layers import Dense, Conv1D, Embedding, GlobalAveragePooling1D
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt
```

Load the dataset.

The screenshot shows a Jupyter Notebook interface with a dark theme. The Explorer panel on the left shows a file named 'movie.csv' under the 'ASSIGNMENT-2' folder. The main editor area displays the first cell of code, which is titled 'Load Dataset.' and contains the following Python code:

```
[7]: data = pd.read_csv("movie.csv")
data.dropna(inplace=True)
print(data.describe())
print(data.head())
```

Below the code, the output of the `data.describe()` function is displayed, showing statistical information for the 'label' column and a preview of the first five rows of the dataset:

	label
count	40000.000000
mean	0.499525
std	0.500006
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

The notebook interface also shows the 'OUTLINE' and 'TIMELINE' panels at the bottom left, and the status bar at the bottom right indicating 'Spaces: 4' and 'Cell 1 of 13'.

Preprocess and split the data.

The screenshot shows the same Jupyter Notebook interface, but now displaying the second cell of code, which is titled 'Preprocess and split data.' and contains the following Python code:

```
[9]: text = data["text"].str.lower()
y = data["label"]

# Define the vocabulary size and embedding dimension
vocab_size = 10000
embedding_dim = 128
max_len = 100

# Tokenize the text data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(text)
sequences = tokenizer.texts_to_sequences(text)

# Pad sequences to ensure uniform length
padded_sequences = pad_sequences(sequences, maxlen=max_len)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, y, test_size=0.3, random_state=42)

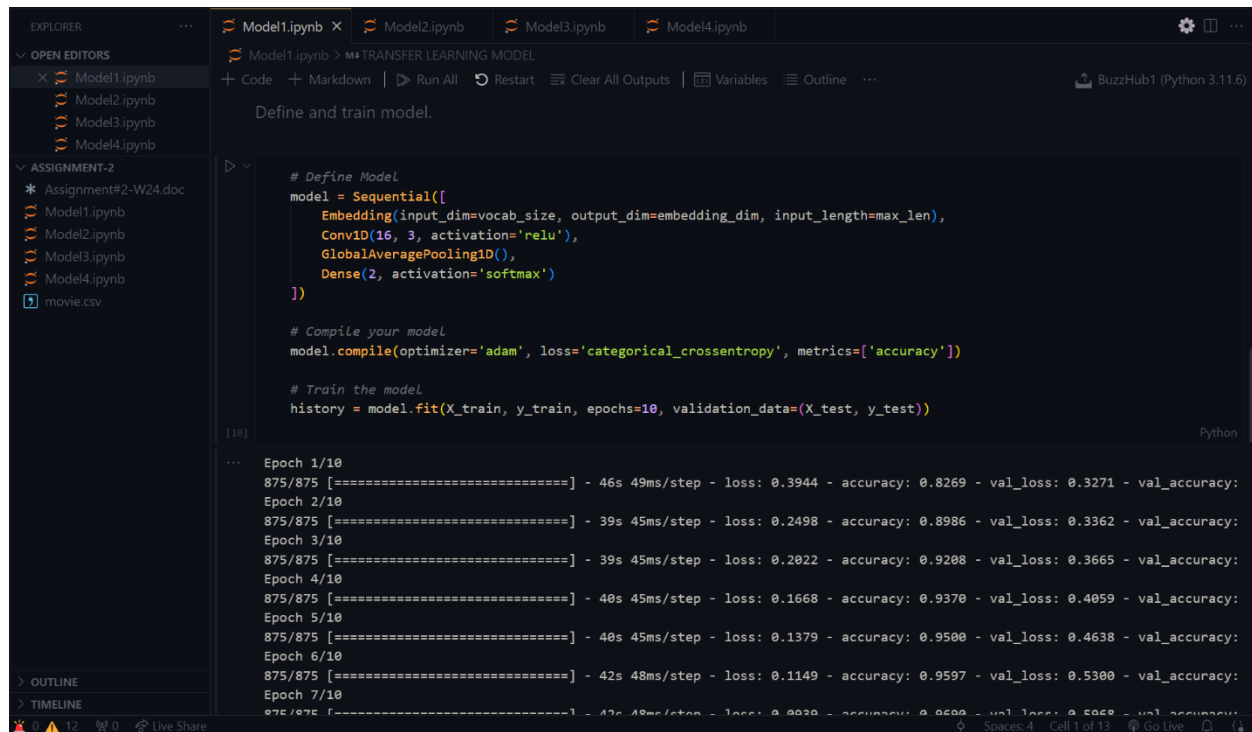
# One-hot encode the target labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Below the code, the output of the `train_test_split` function is displayed, showing the shapes of the training and testing sets:

```
X_train: (32000, 100)
X_test: (8000, 100)
y_train: (32000, 2)
y_test: (8000, 2)
```

The notebook interface also shows the 'OUTLINE' and 'TIMELINE' panels at the bottom left, and the status bar at the bottom right indicating 'Spaces: 4' and 'Cell 1 of 13'.

Define and train the model.



The Jupyter Notebook interface shows the Explorer panel on the left with files Model1.ipynb, Model2.ipynb, Model3.ipynb, Model4.ipynb, and movie.csv. The main editor displays the code for defining and training a model. The code defines a Sequential model with an Embedding layer, a Conv1D layer, a GlobalAveragePooling1D layer, and a Dense layer. It then compiles the model with the Adam optimizer and categorical crossentropy loss, and trains it for 10 epochs. The output shows the training progress for each epoch, including loss, accuracy, and validation loss and accuracy.

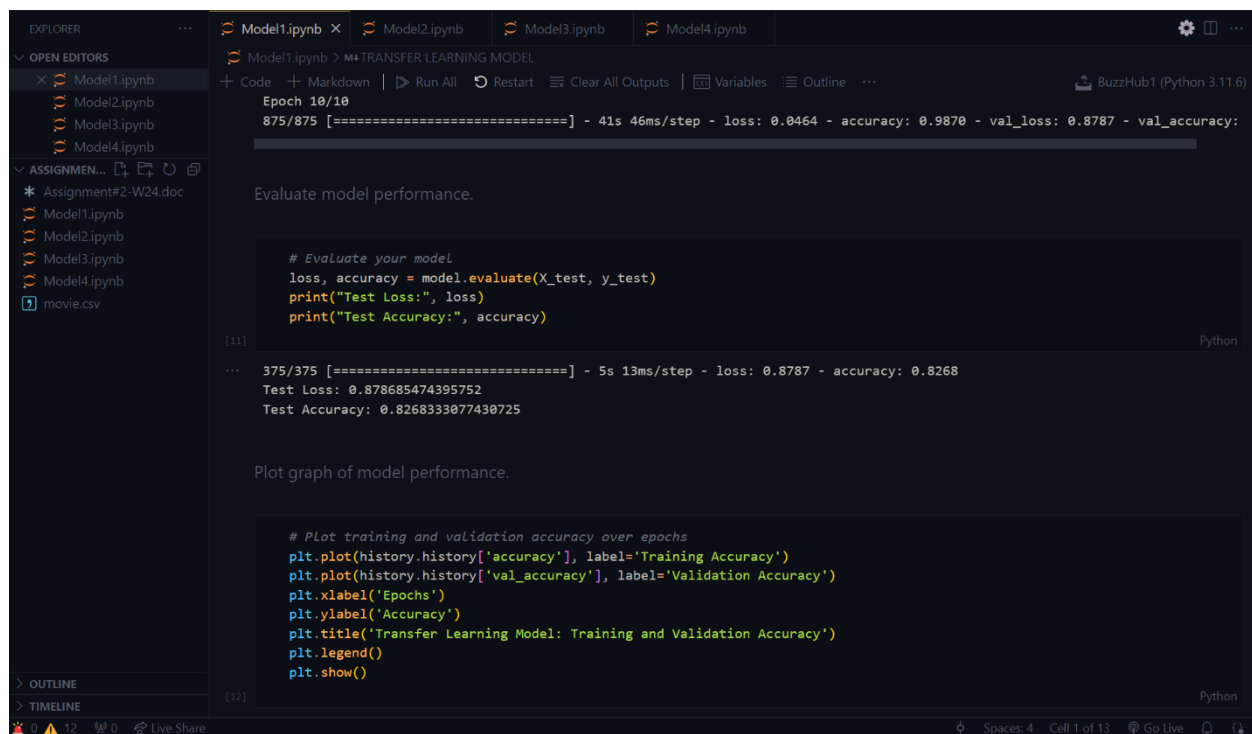
```
# Define Model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_len),
    Conv1D(16, 3, activation='relu'),
    GlobalAveragePooling1D(),
    Dense(2, activation='softmax')
])

# Compile your model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

Epoch 1/10
875/875 [=====] - 46s 49ms/step - loss: 0.3944 - accuracy: 0.8269 - val_loss: 0.3271 - val_accuracy:
Epoch 2/10
875/875 [=====] - 39s 45ms/step - loss: 0.2498 - accuracy: 0.8986 - val_loss: 0.3362 - val_accuracy:
Epoch 3/10
875/875 [=====] - 39s 45ms/step - loss: 0.2022 - accuracy: 0.9208 - val_loss: 0.3665 - val_accuracy:
Epoch 4/10
875/875 [=====] - 40s 45ms/step - loss: 0.1668 - accuracy: 0.9370 - val_loss: 0.4059 - val_accuracy:
Epoch 5/10
875/875 [=====] - 40s 45ms/step - loss: 0.1379 - accuracy: 0.9500 - val_loss: 0.4638 - val_accuracy:
Epoch 6/10
875/875 [=====] - 42s 48ms/step - loss: 0.1149 - accuracy: 0.9597 - val_loss: 0.5300 - val_accuracy:
Epoch 7/10
875/875 [=====] - 42s 48ms/step - loss: 0.0930 - accuracy: 0.9690 - val_loss: 0.5669 - val_accuracy:

Make predictions and evaluate model performance.



The Jupyter Notebook interface shows the Explorer panel on the left with files Model1.ipynb, Model2.ipynb, Model3.ipynb, Model4.ipynb, and movie.csv. The main editor displays the code for evaluating the model performance. The code evaluates the model on the test set, prints the test loss and accuracy, and plots the training and validation accuracy over epochs. The output shows the evaluation results for the test set and the plot of training and validation accuracy over epochs.

```
# Evaluate your model
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

Epoch 10/10
875/875 [=====] - 41s 46ms/step - loss: 0.0464 - accuracy: 0.9870 - val_loss: 0.8787 - val_accuracy:

Evaluate model performance.

```
# Plot training and validation accuracy over epochs
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Transfer Learning Model: Training and Validation Accuracy')
plt.legend()
plt.show()
```

375/375 [=====] - 5s 13ms/step - loss: 0.8787 - accuracy: 0.8268
Test Loss: 0.878685474395752
Test Accuracy: 0.8268333077430725

Plot graph of model performance.

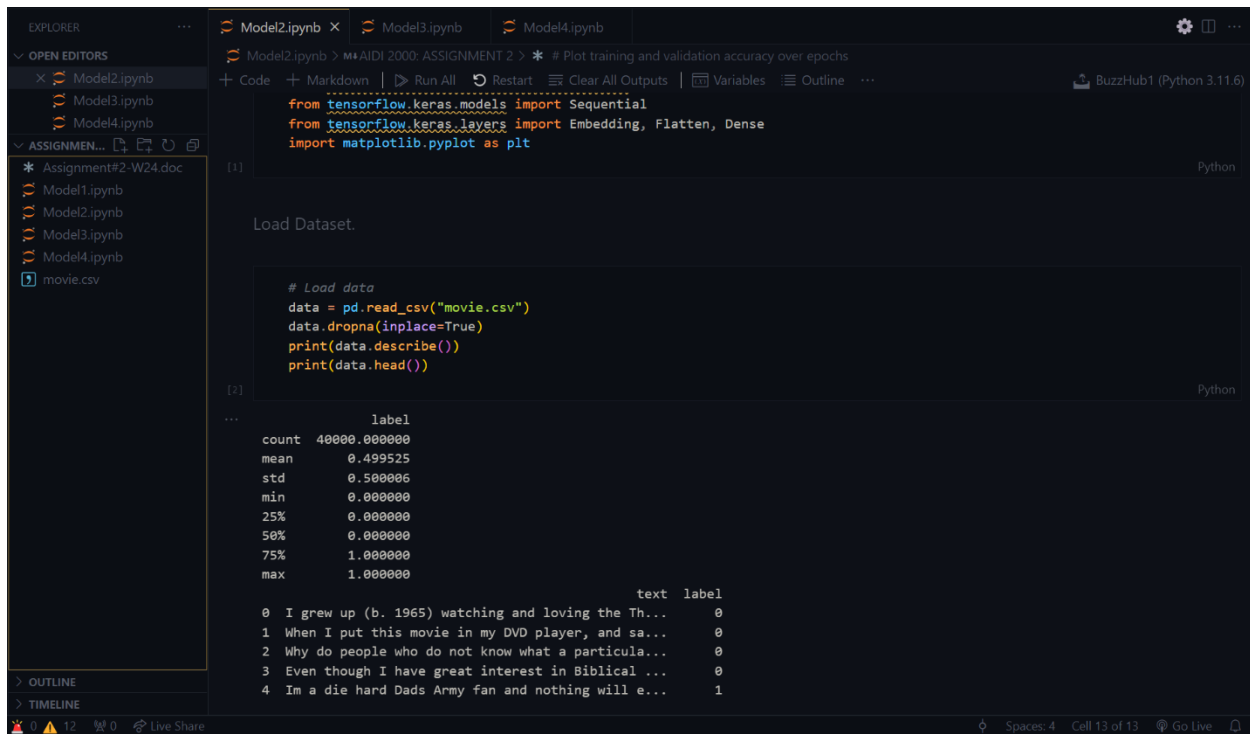


Model 2: Word Embedding.

Load necessary libraries.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense
import matplotlib.pyplot as plt
```

Load dataset.



```
Model2.ipynb > MA AIDI 2000: ASSIGNMENT 2 > * # Plot training and validation accuracy over epochs
+ Code + Markdown | ▶ Run All ⌂ Restart ⌂ Clear All Outputs | 📄 Variables 📄 Outline ... BuzzHub1 (Python 3.11.6)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense
import matplotlib.pyplot as plt

(1)

Python

Load Dataset.

# Load data
data = pd.read_csv("movie.csv")
data.dropna(inplace=True)
print(data.describe())
print(data.head())

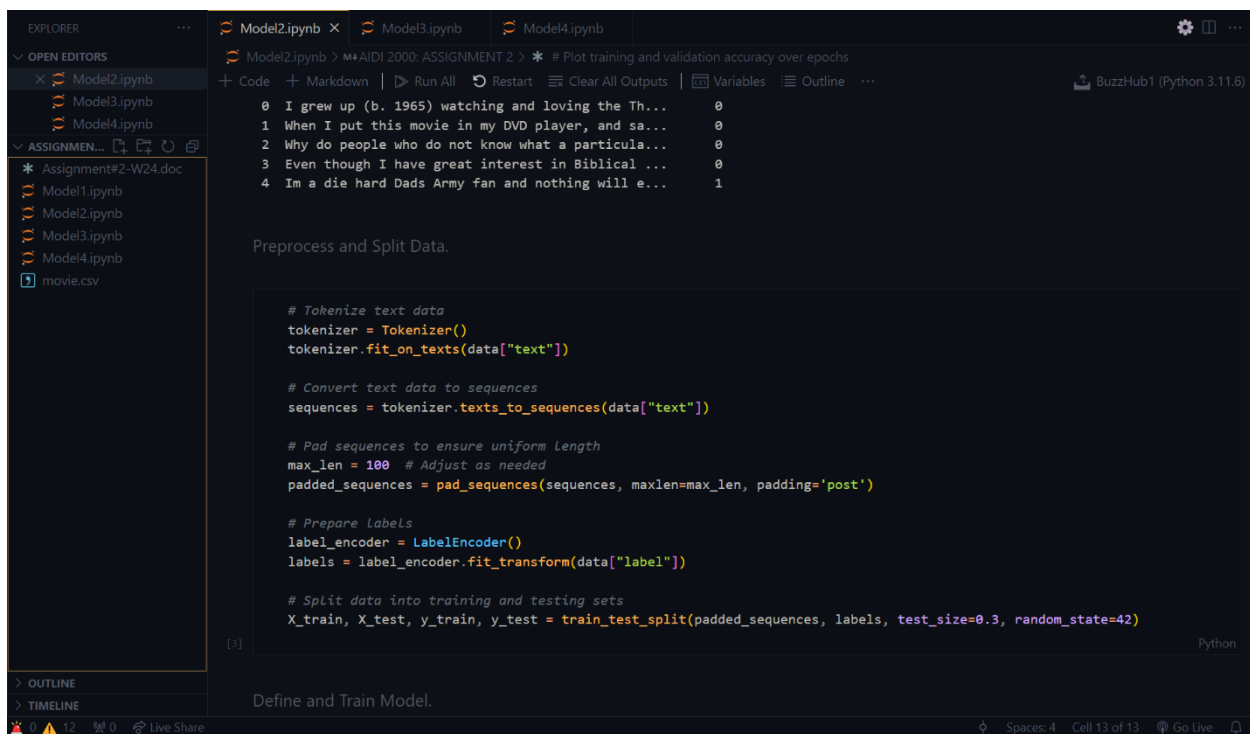
(2)

Python

...
      label
count  40000.000000
mean    0.499525
std      0.500006
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      1.000000

      text  label
0  I grew up (b. 1965) watching and loving the Th...    0
1  When I put this movie in my DVD player, and sa...    0
2  Why do people who do not know what a particula...    0
3  Even though I have great interest in Biblical ...    0
4  Im a die hard Dads Army fan and nothing will e...    1
```

Preprocess and split data.



```
Model2.ipynb > MA AIDI 2000: ASSIGNMENT 2 > * # Plot training and validation accuracy over epochs
+ Code + Markdown | ▶ Run All ⌂ Restart ⌂ Clear All Outputs | 📄 Variables 📄 Outline ... BuzzHub1 (Python 3.11.6)

0  I grew up (b. 1965) watching and loving the Th...    0
1  When I put this movie in my DVD player, and sa...    0
2  Why do people who do not know what a particula...    0
3  Even though I have great interest in Biblical ...    0
4  Im a die hard Dads Army fan and nothing will e...    1

Preprocess and Split Data.

# Tokenize text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data["text"])

# Convert text data to sequences
sequences = tokenizer.texts_to_sequences(data["text"])

# Pad sequences to ensure uniform length
max_len = 100 # Adjust as needed
padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post')

# Prepare Labels
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(data["label"])

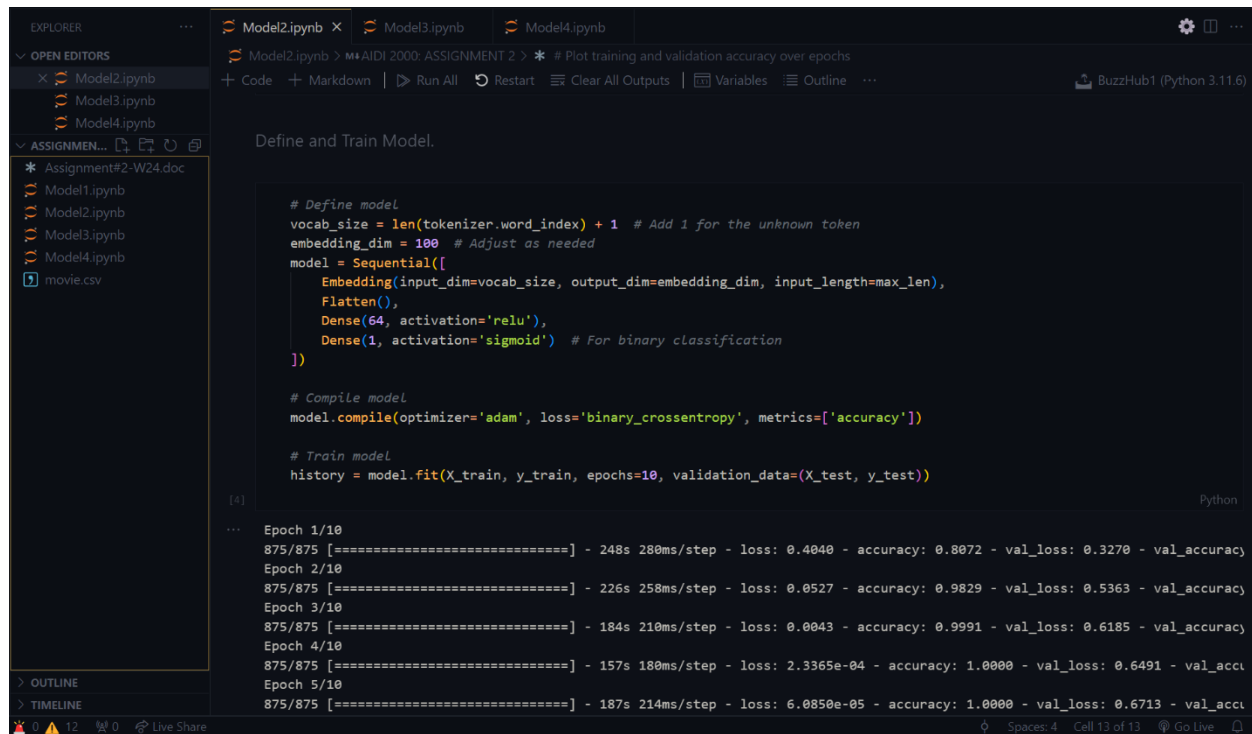
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels, test_size=0.3, random_state=42)

(3)

Python

Define and Train Model.
```

Define and train the model.



The Jupyter Notebook interface shows the Explorer panel on the left with files: Model1.ipynb, Model2.ipynb, Model3.ipynb, Model4.ipynb, movie.csv, and Assignment#2-W24.doc. The main editor displays the code for defining and training a model. The code defines a Sequential model with an Embedding layer, a Flatten layer, and two Dense layers. It then compiles the model with the Adam optimizer and binary crossentropy loss, and trains it for 10 epochs. The output shows the training progress for each epoch, including loss, accuracy, and validation loss and accuracy.

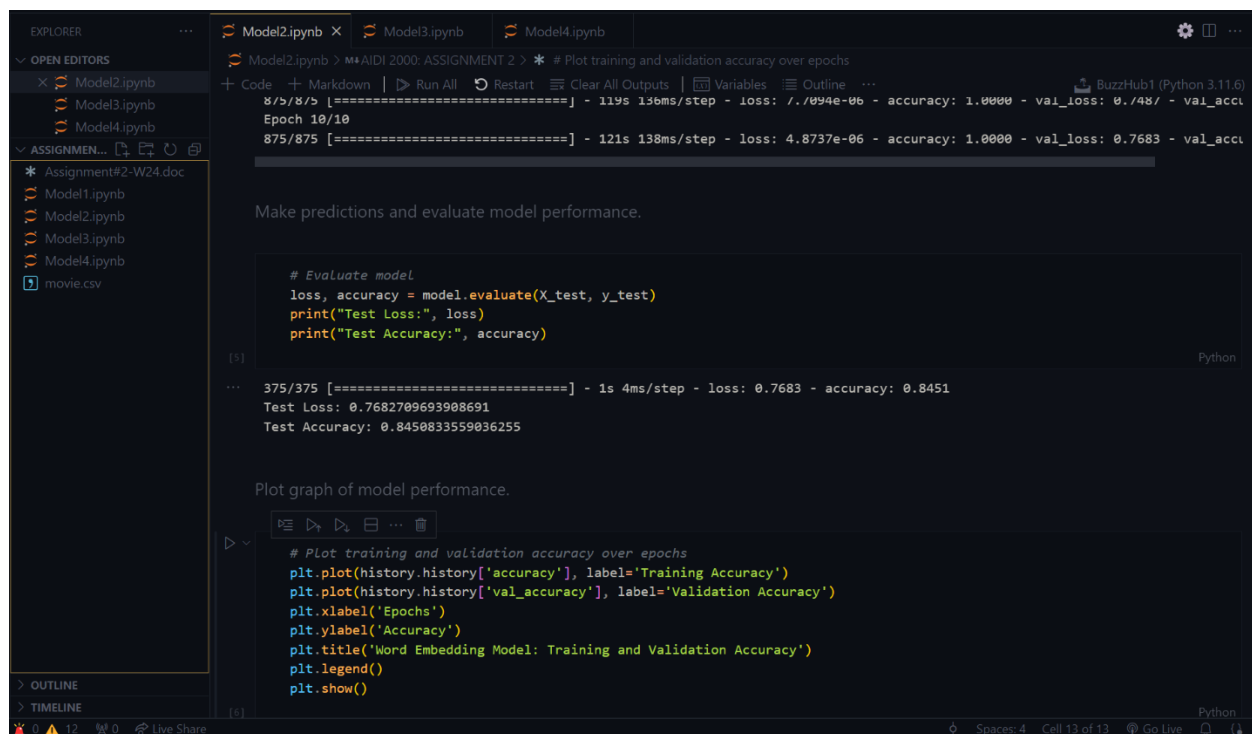
```
# Define model
vocab_size = len(tokenizer.word_index) + 1 # Add 1 for the unknown token
embedding_dim = 100 # Adjust as needed
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_len),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid') # For binary classification
])

# Compile model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train model
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

Epoch 1/10
875/875 [=====] - 248s 280ms/step - loss: 0.4040 - accuracy: 0.8072 - val_loss: 0.3270 - val_accuracy: 0.8072
Epoch 2/10
875/875 [=====] - 226s 258ms/step - loss: 0.0527 - accuracy: 0.9829 - val_loss: 0.5363 - val_accuracy: 0.9829
Epoch 3/10
875/875 [=====] - 184s 210ms/step - loss: 0.0043 - accuracy: 0.9991 - val_loss: 0.6185 - val_accuracy: 0.9991
Epoch 4/10
875/875 [=====] - 157s 180ms/step - loss: 2.3365e-04 - accuracy: 1.0000 - val_loss: 0.6491 - val_accuracy: 1.0000
Epoch 5/10
875/875 [=====] - 187s 214ms/step - loss: 6.0850e-05 - accuracy: 1.0000 - val_loss: 0.6713 - val_accuracy: 1.0000

Make predictions and evaluate model performance.

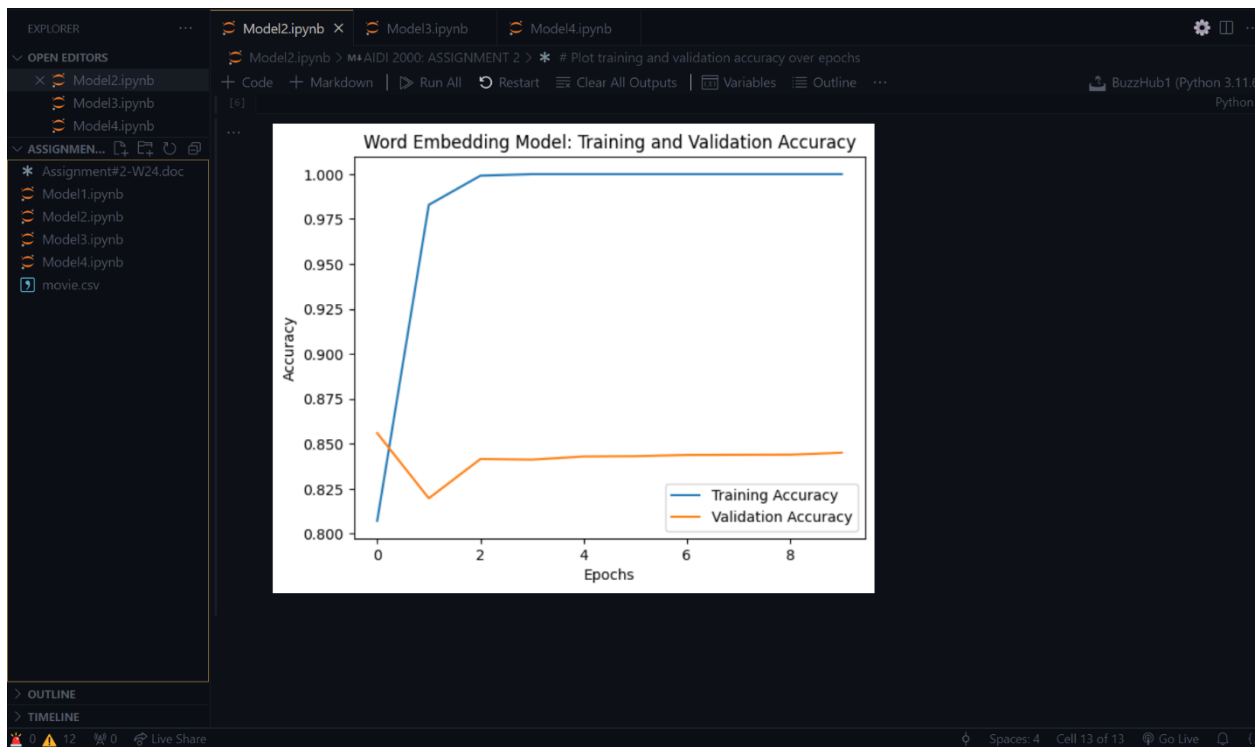


The Jupyter Notebook interface shows the Explorer panel on the left with files: Model1.ipynb, Model2.ipynb, Model3.ipynb, Model4.ipynb, movie.csv, and Assignment#2-W24.doc. The main editor displays the code for evaluating the model and plotting its performance. The code evaluates the model on the test set, printing the test loss and accuracy. It then plots the training and validation accuracy over epochs.

```
# Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

375/375 [=====] - 1s 4ms/step - loss: 0.7683 - accuracy: 0.8451
Test Loss: 0.7682709693908691
Test Accuracy: 0.8450833559036255

```
# Plot training and validation accuracy over epochs
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Word Embedding Model: Training and Validation Accuracy')
plt.legend()
plt.show()
```

Model 3: Logistic Regression.

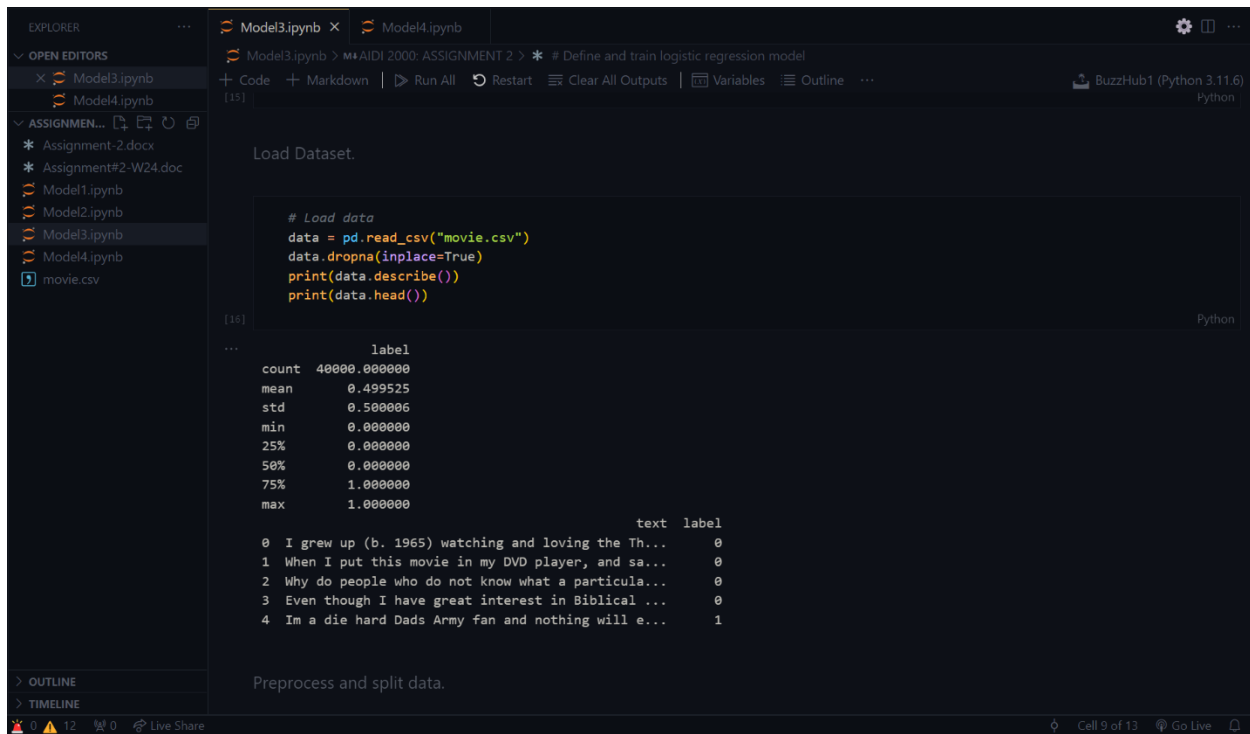
Import necessary libraries.

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

```

Load the dataset.



The screenshot shows a Jupyter Notebook interface with the Explorer panel on the left and the Code editor on the right. The Explorer panel shows a file named 'movie.csv' selected. The Code editor contains the following code:

```
# Load data
data = pd.read_csv("movie.csv")
data.dropna(inplace=True)
print(data.describe())
print(data.head())
```

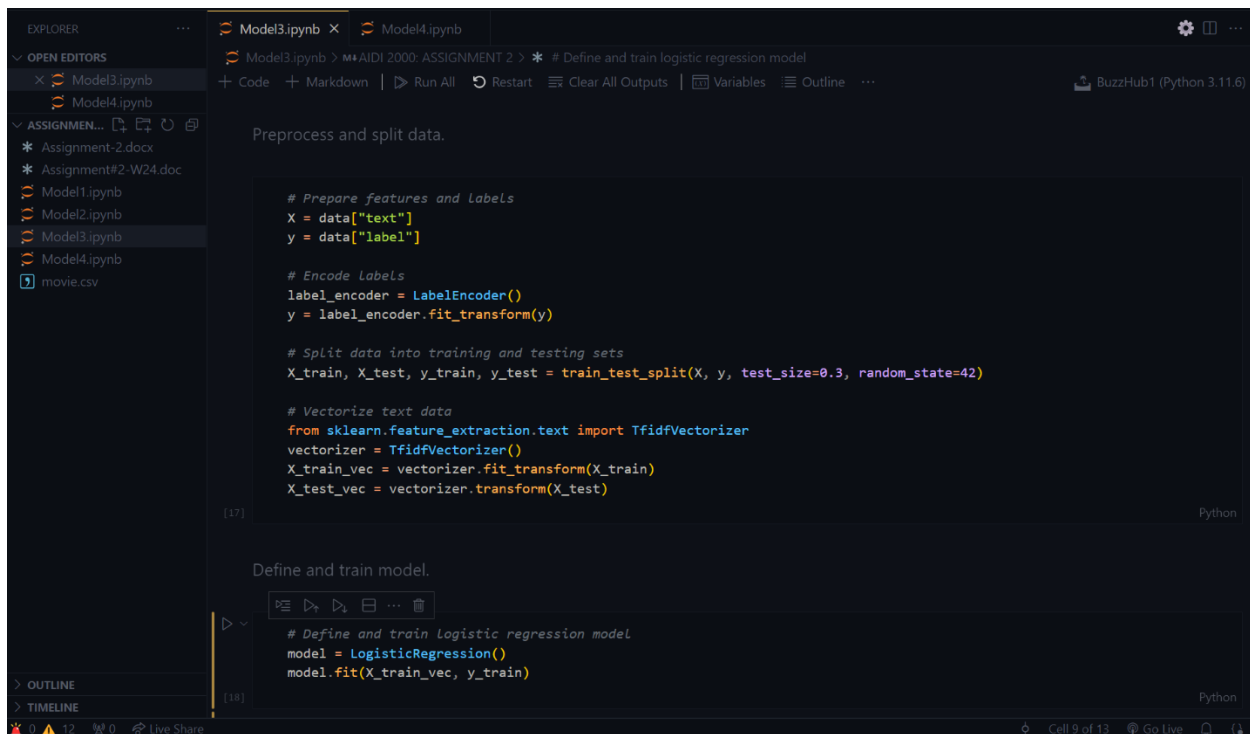
The output of the code is displayed below the code cell:

```
count    40000.000000
mean      0.499525
std       0.500006
min       0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max       1.000000
```

The output also shows a preview of the data:

```
text label
0  I grew up (b. 1965) watching and loving the Th...  0
1  When I put this movie in my DVD player, and sa...  0
2  Why do people who do not know what a particula...  0
3  Even though I have great interest in Biblical ...  0
4  Im a die hard Dads Army fan and nothing will e...  1
```

Preprocess data, split data, define the model and train the model.



The screenshot shows the same Jupyter Notebook interface as the previous one, but with the second cell of code executed. The Code editor contains the following code:

```
# Prepare features and labels
X = data["text"]
y = data["label"]

# Encode labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

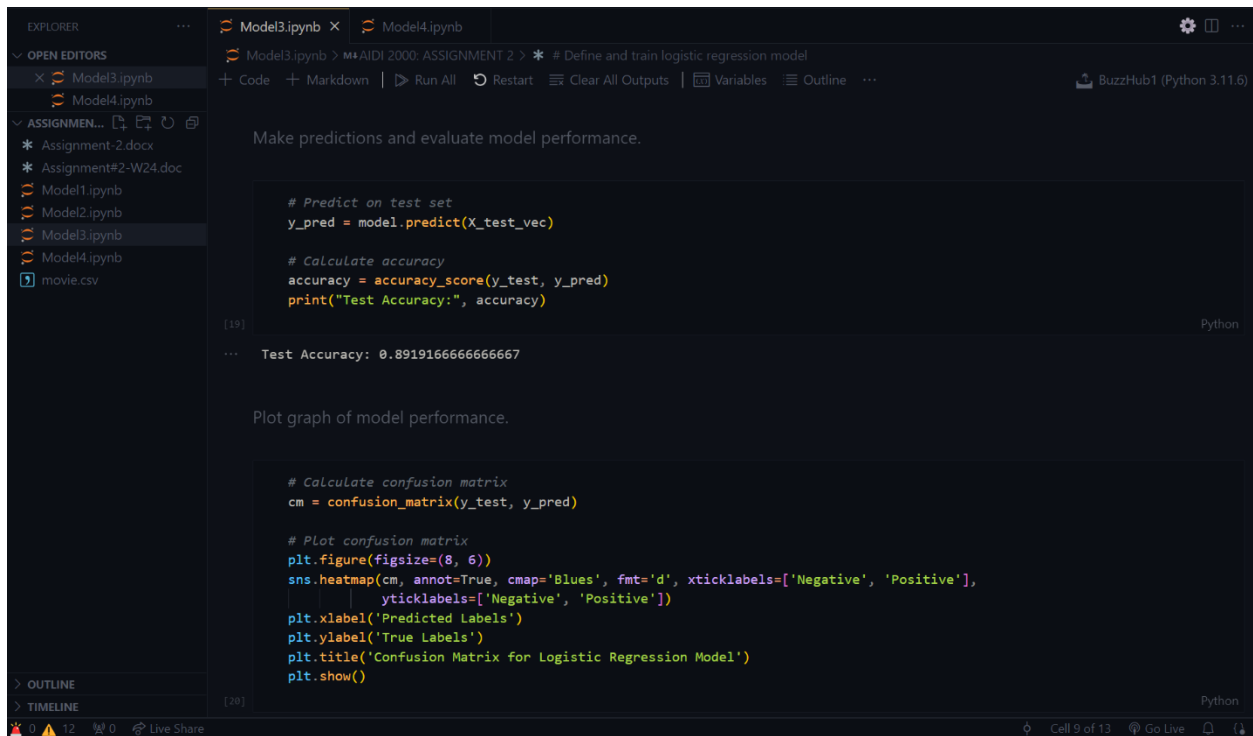
# Vectorize text data
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

The output of the code is displayed below the code cell:

```
Define and train model.
```

```
# Define and train Logistic regression model
model = LogisticRegression()
model.fit(X_train_vec, y_train)
```

Make predictions and evaluate model performance.



The image shows a Jupyter Notebook interface with two open files: Model3.ipynb and Model4.ipynb. The active cell in Model3.ipynb contains the following Python code:

```
# Predict on test set
y_pred = model.predict(X_test_vec)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

The output of the code is:

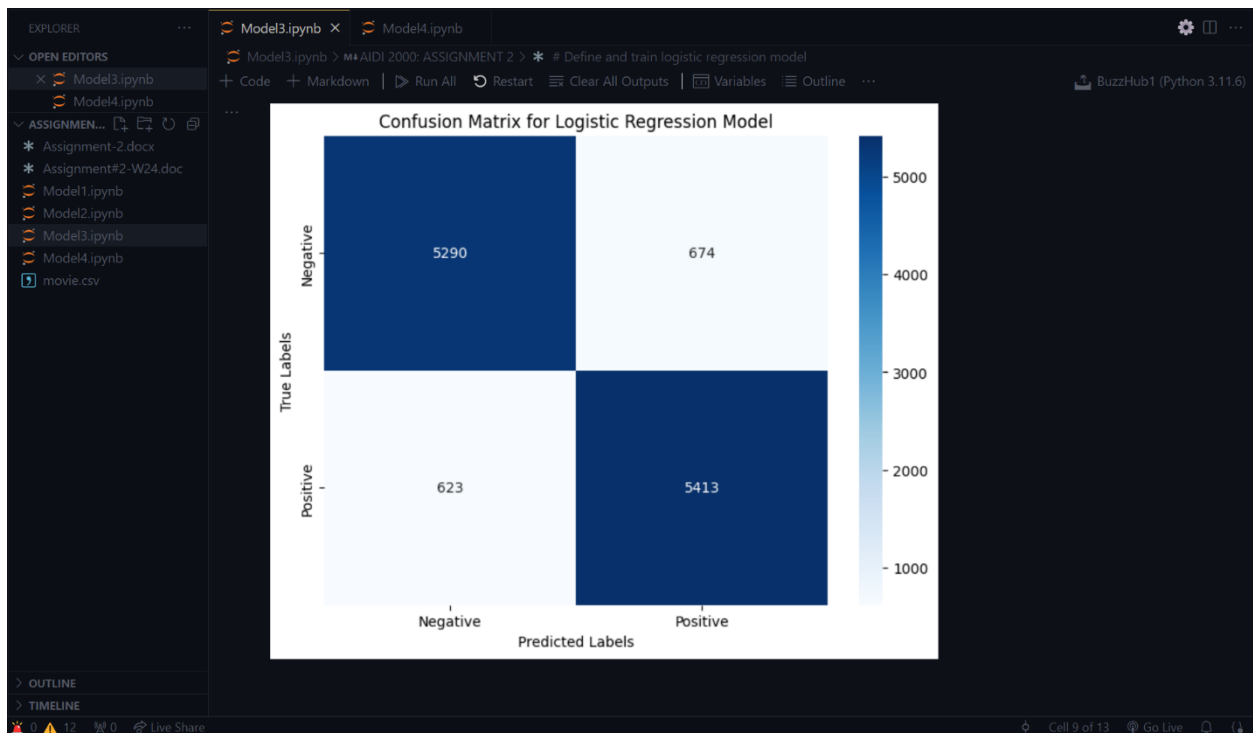
```
Test Accuracy: 0.8919166666666667
```

Below the code, there is a section titled "Plot graph of model performance." which contains the following Python code:

```
# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

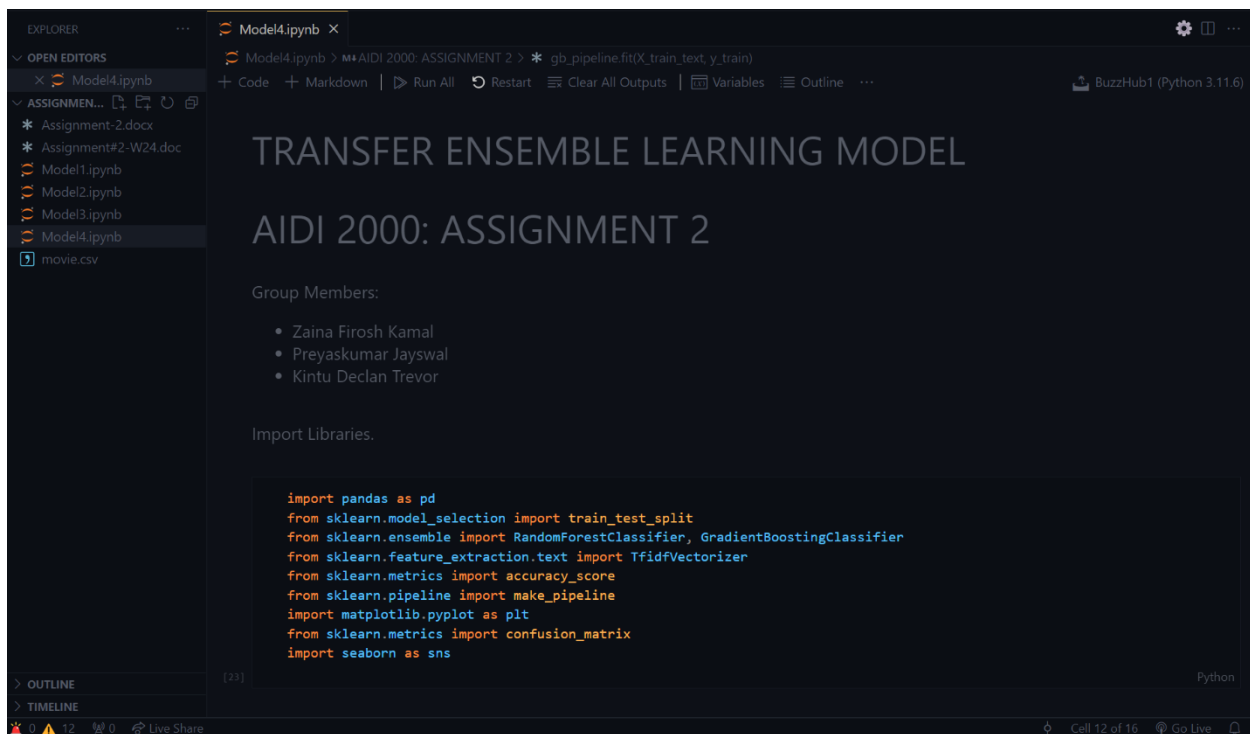
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Logistic Regression Model')
plt.show()
```

The output of the code is a confusion matrix plot.



Model 4: Ensemble Learning.

Load the Necessary Libraries.



Model4.ipynb

Model4.ipynb > AIDI 2000: ASSIGNMENT 2 > gb_pipeline.fit(X_train_text, y_train)

Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline

BuzzHub1 (Python 3.11.6)

TRANSFER ENSEMBLE LEARNING MODEL

AIDI 2000: ASSIGNMENT 2

Group Members:

- Zaina Firosh Kamal
- Preyaskumar Jayswal
- Kintu Declan Trevor

Import Libraries.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

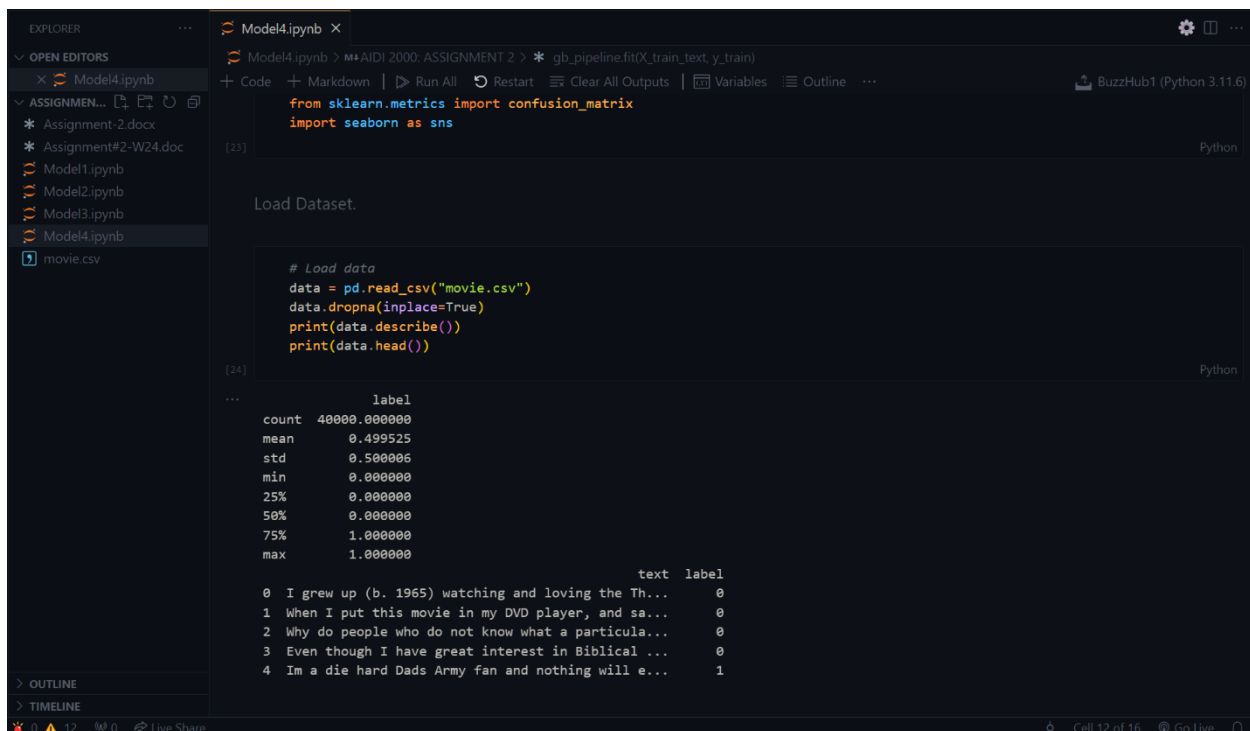
[23]

Python

0 12 0 Live Share

Cell 12 of 16 Go Live

Load dataset.



Model4.ipynb

Model4.ipynb > AIDI 2000: ASSIGNMENT 2 > gb_pipeline.fit(X_train_text, y_train)

Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline

BuzzHub1 (Python 3.11.6)

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

[23]

Python

Load Dataset.

```
# Load data
data = pd.read_csv("movie.csv")
data.dropna(inplace=True)
print(data.describe())
print(data.head())
```

[24]

Python

```
count    40000.000000
mean      0.499525
std       0.500006
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

0 12 0 Live Share

Cell 12 of 16 Go Live

Preprocess data, split data, define base models and train models.

The screenshot shows a Jupyter Notebook with two code cells. The first cell, labeled [25], contains code for preprocessing and splitting data. The second cell, labeled [26], contains code for defining and training base models. The notebook interface includes an Explorer pane on the left, a toolbar at the top, and a status bar at the bottom.

```
Model4.ipynb > MA AIDI 2000: ASSIGNMENT 2 > * gb_pipeline.fit(X_train_text, y_train)
+ Code + Markdown | ▶ Run All ⏮ Restart ⏻ Clear All Outputs | 📄 Variables 📄 Outline ...
4 I'm a die hard Dads Army fan and nothing will e... 1

Preprocess and split data.

# Prepare features and labels
X_text = data["text"]
y = data["label"]

# Split data into training and testing sets
X_train_text, X_test_text, y_train, y_test = train_test_split(X_text, y, test_size=0.3, random_state=42)

[25] Python

Define and train models.

# Define TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=10000) # Adjust max_features as needed

# Define base models
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)

# Create pipelines for base models
rf_pipeline = make_pipeline(tfidf_vectorizer, rf_model)
gb_pipeline = make_pipeline(tfidf_vectorizer, gb_model)

[26] Python
```

The screenshot shows a Jupyter Notebook with two code cells. The first cell, labeled [27], contains the code `rf_pipeline.fit(X_train_text, y_train)`. The second cell, labeled [28], contains the code `gb_pipeline.fit(X_train_text, y_train)`. Both cells show the resulting pipeline objects as visualizations. The notebook interface includes an Explorer pane on the left, a toolbar at the top, and a status bar at the bottom.

```
Model4.ipynb > MA AIDI 2000: ASSIGNMENT 2 > * gb_pipeline.fit(X_train_text, y_train)
+ Code + Markdown | ▶ Run All ⏮ Restart ⏻ Clear All Outputs | 📄 Variables 📄 Outline ...
Train Base Models

rf_pipeline.fit(X_train_text, y_train)

[27] Python

Pipeline
Pipeline(steps=[('tfidfvectorizer', TfidfVectorizer(max_features=10000)),
                ('randomforestclassifier', RandomForestClassifier(random_state=42))])
├── TfIdfVectorizer
│   └── TfIdfVectorizer(max_features=10000)
└── RandomForestClassifier
    └── RandomForestClassifier(random_state=42)

gb_pipeline.fit(X_train_text, y_train)

[28] Python

Pipeline
Pipeline(steps=[('tfidfvectorizer', TfidfVectorizer(max_features=10000)),
                ('gradientboostingclassifier', GradientBoostingClassifier(random_state=42))])
├── TfIdfVectorizer
│   └── TfIdfVectorizer(max_features=10000)
└── GradientBoostingClassifier
    └── GradientBoostingClassifier(random_state=42)
```

Make predictions and evaluate model performance.

```
EXPLORER
Model4.ipynb
Model4.ipynb
Model1.ipynb
Model2.ipynb
Model3.ipynb
Model4.ipynb
movie.csv

Model4.ipynb
Model4.ipynb > M4AIDI 2000: ASSIGNMENT 2 > * gb_pipeline.fit(X_train_text, y_train)
+ Code + Markdown | ▶ Run All ⏹ Restart ⌵ Clear All Outputs | 📄 Variables 📄 Outline ...
BuzzHub1 (Python 3.11.6)

# Make predictions
rf_pred = rf_pipeline.predict(X_test_text)
gb_pred = gb_pipeline.predict(X_test_text)

# Ensemble predictions
ensemble_pred = (rf_pred + gb_pred) / 2 # Simple averaging for binary classification
ensemble_pred_binary = ensemble_pred.round()

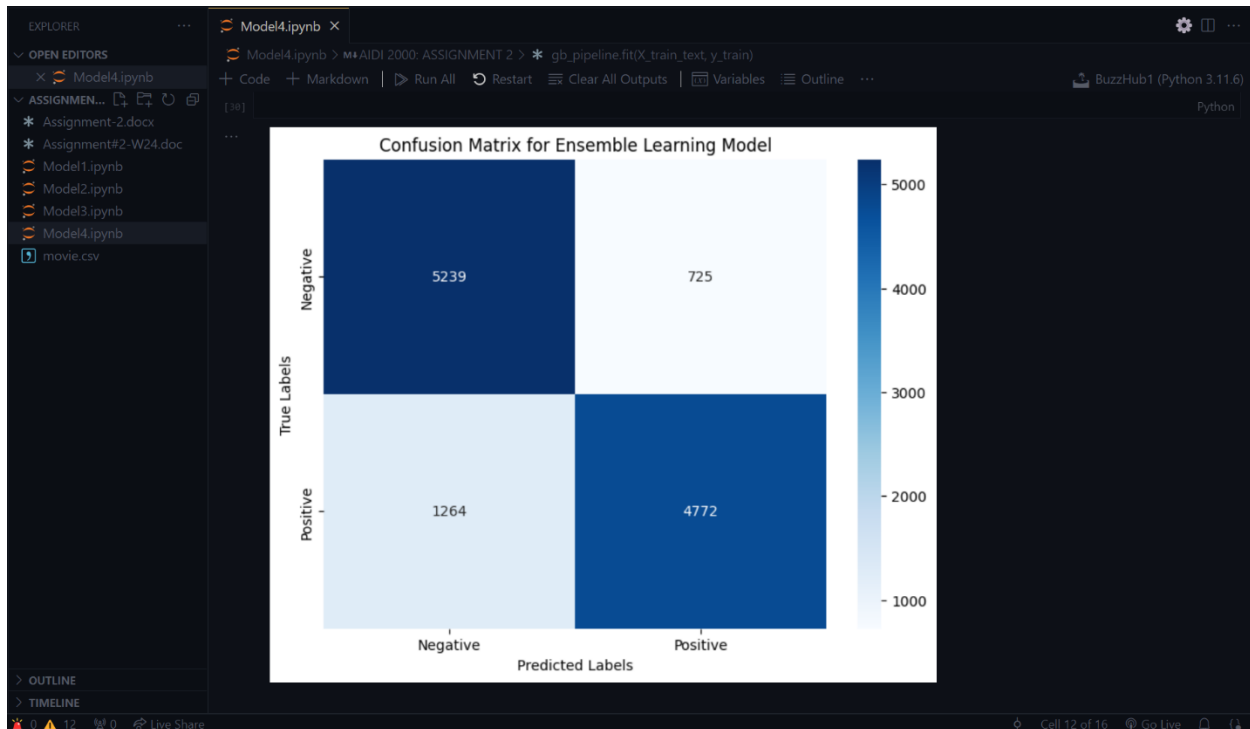
# Calculate accuracy
accuracy = accuracy_score(y_test, ensemble_pred_binary)
print("Ensemble Model Accuracy:", accuracy)

[29] Python
... Ensemble Model Accuracy: 0.83425

Plot graph of model performance.

# Calculate confusion matrix
cm = confusion_matrix(y_test, ensemble_pred_binary)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Ensemble Learning Model')
plt.show()
```



OUR RECOMMENDATION:

Based on our evaluation of all four models, we found Model 3 (Logistic Regression) to be the most successful in making predictions about the dataset.