جهت سهولت در نوشتن کد ها، هر فاز به صورت جداگانه نوشته شده و در نهایت خروجی فاز برای استفاده در مراحل بعدی ذخیره شده.

کتابخانه های استفاده شده:

1. import pandas as pd

2. from sklearn.model\_selection import train\_test\_split

3. from textblob import TextBlob

4. import matplotlib.pyplot as plt

5. from nltk.stem import PorterStemmer

6. import re

7.

8. from nltk.corpus import stopwords

9. from sklearn.feature\_extraction.text import CountVectorizer

10. from sklearn.cluster import KMeans

11. from sklearn.manifold import TSNE

12. import matplotlib.pyplot as plt

13. import numpy as np

14.

15. from tensorflow.keras.preprocessing.text import Tokenizer

16. from tensorflow.keras.preprocessing.sequence import pad\_sequences

17.

18. import urllib.request

19. import zipfile

20. import os

21.

22. from tensorflow.keras.models import Sequential

23. from tensorflow.keras.layers import Embedding, GRU, Dense, Bidirectional

24. from tensorflow.keras.callbacks import History

1-پیش پردازش

در مرحله اول یک تابع برای بازکردن فایل اکسل نوشته شده:

1. # Step 1: Load the dataset

2. def load\_dataset(file\_path):

3. df = pd.read\_csv(file\_path)

4. return df

مرحله 1-1-

در این مرحله مثبت ها را به یک تبدیل کردیم و منفی ها را به صفر

1. # Step 1.1: Convert positive labels to 1 and negative labels to 0

2. def convert\_labels(df):

3. df['sentiment'] = df['sentiment'].apply(lambda x: 1 if x == 'positive' else 0)

4. return df

مرحله 1-2-

در این مرحله رکود های تکراری حذف شده. از inplace استفاده شده تا فایل جدیدی ایجاد نشود و در همان فایل تغییرات اعمال شود.

1. # Step 1.2: Remove duplicate rows

2. def remove\_duplicates(df):

3. df.drop\_duplicates(inplace=True)

مرحله 1-3-

در این مرحله 30 درصد داده ها برای آموزش با رندم استیت 42 جدا شده اند:

1. # Step 1.3: Randomly select 30% of the dataset

2. def random\_selection(df):

3. df = df.sample(frac=0.3, random\_state=42)

4. return df

مرحله 1-4-

در این مرحله یک ستون جدید ایجاد شده و با استفاده از کتابخانه textBlob قطبیت جمله در آن ثبت شده:

1. # Step 1.4: Calculate polarity using TextBlob

2. def get\_polarity(df):

3. print('step-1-4')

4. df['polarity'] = df['review'].apply(lambda x: TextBlob(str(x)).sentiment.polarity)

5. return df

مرحله 1-5-

در این مرحله از ستون sentiment ستون های 1 را مثبت و 0 را منفی در نظر گرفتیم و با پلات نمودار رسم کرده و ذخیره کردیم:

1. # Step 1.5: Separate positive and negative polarities, plot histograms, and analyze means

2. def plot\_sentiment\_histograms(df):

3. print('step-1-5')

4. positive\_reviews = df[df['sentiment'] == 1]

5. negative\_reviews = df[df['sentiment'] == 0]

6.

7. # Plotting histogram for positive sentiment

8. plt.hist(positive\_reviews['polarity'], bins=20, color='green', alpha=0.7)

9. plt.title('positive')

10. plt.xlabel('sense')

11. plt.ylabel('count')

12. plt.savefig('positive.png')

13. plt.show()

14.

15. # Plotting histogram for negative sentiment

16. plt.hist(negative\_reviews['polarity'], bins=20, color='red', alpha=0.7)

17. plt.title('negative')

18. plt.xlabel('sense')

19. plt.ylabel('count')

20. plt.savefig('negative.png')

21. plt.show()

22.

23. # Calculate mean polarity for positive and negative reviews

24. mean\_polarity\_positive = positive\_reviews['polarity'].mean()

25. mean\_polarity\_negative = negative\_reviews['polarity'].mean()

26.

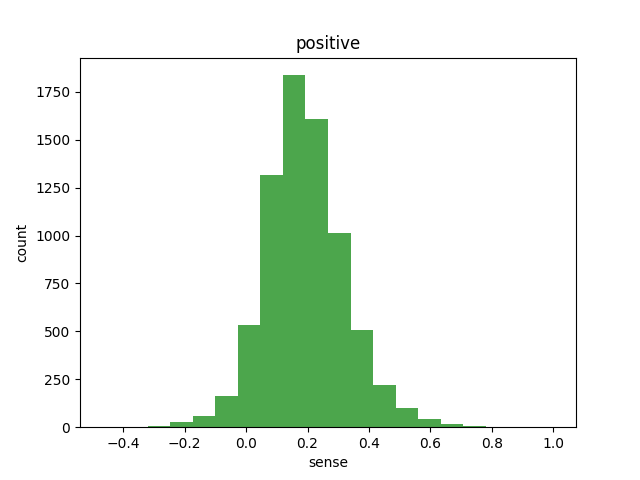
27. # Print mean polarities

28. print("Mean Polarity for Positive Reviews:", mean\_polarity\_positive)

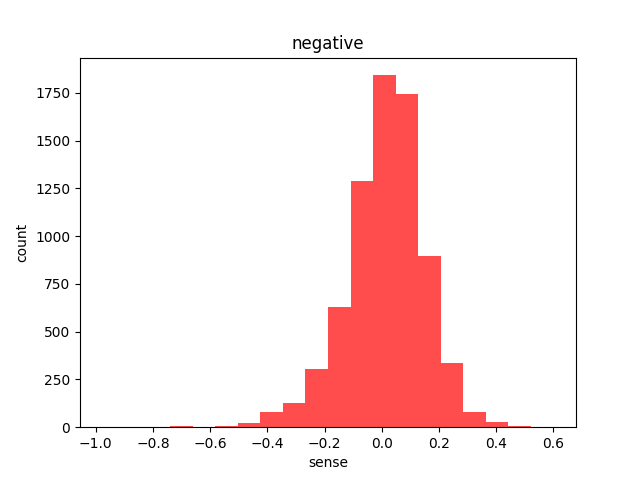
29. print("Mean Polarity for Negative Reviews:", mean\_polarity\_negative)

30.

خروجی مثبت:



خروجی منفی:



خروجی میانگین ها در کنسول:

1. Mean Polarity for Positive Reviews: 0.19375026016013788

2. Mean Polarity for Negative Reviews: 0.012643524175639971

مرحله 1-6-

در این مرحله ابتدا تمام حروف کوچک شده و با رجکس اصلاح شده اند سپس استاپ ورد ها حذف شده و استم کلمات استخراج شده:

1. # Step 1.6: Apply preprocessing steps

2. def preprocess\_text(text):

3. text = text.lower()

4. text = re.sub(r'\W', ' ', text)

5. text = re.sub(r'\s+', ' ', text)

6.

7. stop\_words = set(stopwords.words('english'))

8. text = ' '.join([word for word in text.split() if word not in stop\_words])

9.

10. stemmer = PorterStemmer()

11. text = ' '.join([stemmer.stem(word) for word in text.split()])

مرحله 1-7-

برای تقسیم داده به شکر زیر اقدام شده:

1. # Step 1.7: Split the dataset into training and testing sets

2. def split\_dataset(df):

3. train\_data, test\_data, train\_labels, test\_labels = train\_test\_split(

4. df['review'], df['sentiment'], test\_size=0.5, random\_state=42

5. )

6. return train\_data, test\_data, train\_labels, test\_labels

2-خوشه بندی

مرحله 2-1-

در این CountVectorizer فراخوانی شده و داده های آموزشی و تست فیت ترنسفورم شده اند.

1. # Step 2-1: Create Bag of Words (BOW) representation for each review

2. def create\_bow\_representation(reviews\_train, reviews\_test):

3. vectorizer = CountVectorizer()

4.

5. bow\_matrix\_train = vectorizer.fit\_transform(reviews\_train)

6. bow\_matrix\_test = vectorizer.transform(reviews\_test)

7.

8. return bow\_matrix\_train, bow\_matrix\_test

مرحله 2-2-

در این مرحله خوشه بند k-means را فراخوانی کرده و دسته بندی را انجام دادیم:

1. # Step 2-2: Cluster the training and testing sets using K-means

2. def kmeans\_clustering(bow\_matrix\_train, bow\_matrix\_test, num\_clusters=3):

3. kmeans = KMeans(n\_clusters=num\_clusters, random\_state=42)

4.

5. # Fit on training set

6. train\_clusters = kmeans.fit\_predict(bow\_matrix\_train)

7.

8. # Predict on testing set

9. test\_clusters = kmeans.predict(bow\_matrix\_test)

10.

11. return train\_clusters, test\_clusters

12.

مرحله 2-3-

در این مرحله با استفاده tsne خوشه ها رسم شده اند و در پلات نمایش داده شده اند:

1. # Step 2-3: Visualize clusters using t-sne

2. def visualize\_clusters(bow\_matrix, clusters, title):

3. print('step-2-3')

4. tsne = TSNE(n\_components=2, random\_state=42)

5. tsne\_result = tsne.fit\_transform(bow\_matrix.toarray())

6.

7. plt.figure(figsize=(8, 6))

8. scatter = plt.scatter(tsne\_result[:, 0], tsne\_result[:, 1], c=clusters, cmap='viridis')

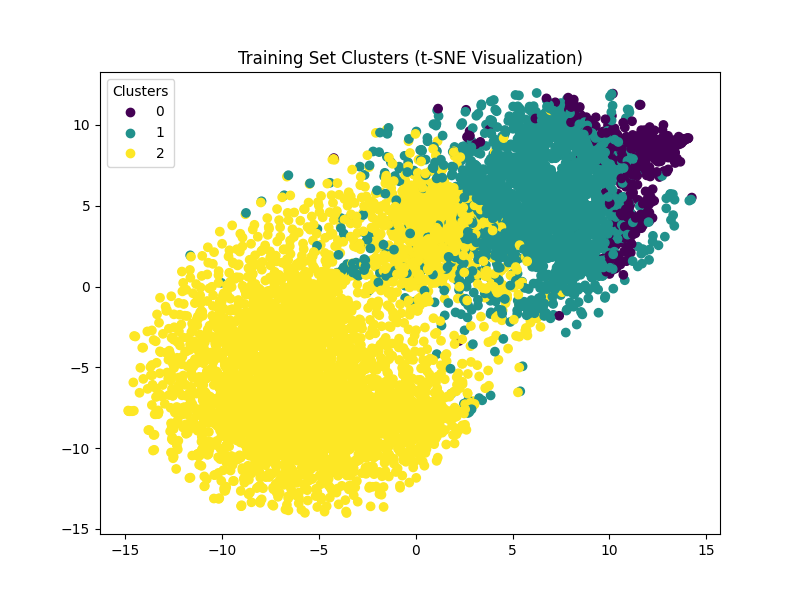
9. plt.title(title)

10. plt.legend(\*scatter.legend\_elements(), title="Clusters")

11. plt.savefig('training\_clusters\_plot.png')

12. plt.show()

خروجی:



مرحله 2-4-

در این مرحله ابتدا از هر دسته داده های یکتا را جدا کردیم و مینیم تعداد را مشخص کردیم و به تعداد مینیمم از داده ها به صورت رندوم انتخاب کردیم، همچنین اگر داده ای تکراری وجود داشته باشد آن را جایگزین نمی کنیم.

1. # Step 2-4: Select a uniform sample from each cluster

2. def select\_uniform\_sample(bow\_matrix, clusters, num\_samples):

3. unique\_clusters, cluster\_counts = np.unique(clusters, return\_counts=True)

4.

5. selected\_samples = []

6.

7. for cluster in unique\_clusters:

8. cluster\_indices = np.where(clusters == cluster)[0]

9. mi = min(num\_samples, cluster\_counts[cluster])

10. selected\_indices = np.random.choice(cluster\_indices, size=mi, replace=False)

11. selected\_samples.extend(selected\_indices)

12.

13. return selected\_samples

مرحله 3 بردار سازی متن

مرحله 3-1-

در این مرحله توکنایز را فراخوانی کرده ایم و بر روی داده های تست و آموزشی آن را اعمال کرده ایم:

1. # Step 3-1: Convert text sequences to numerical sequences using Tokenizer

2. def tokenize\_text\_sequences(X\_train, X\_test):

3. tokenizer = Tokenizer()

4. tokenizer.fit\_on\_texts(X\_train)

5.

6. # Convert text sequences to numerical sequences

7. train\_sequences = tokenizer.texts\_to\_sequences(X\_train)

8. test\_sequences = tokenizer.texts\_to\_sequences(X\_test)

9.

10. return tokenizer, train\_sequences, test\_sequences

مرحله 3-2-

در نهایت ماکسیم گیری انجام شده و به اندازه آن پدینگ اعمال شده:

1. # Step 3-2: Pad sequences to have a consistent length

2. def pad\_text\_sequences(train\_sequences, test\_sequences):

3. max\_length = max(max(len(seq) for seq in train\_sequences), max(len(seq) for seq in test\_sequences))

4.

5. # Pad sequences

6. padded\_train\_sequences = pad\_sequences(train\_sequences, maxlen=max\_length, padding='post')

7. padded\_test\_sequences = pad\_sequences(test\_sequences, maxlen=max\_length, padding='post')

8.

9. return padded\_train\_sequences, padded\_test\_sequences, max\_length

مرحله 4- تعبیه کلمات

در این مرحله طبق کد نمونه فایل مورد نظر دانلود شده و سپس امبدنیگ بر روی کلمات اعمال شده تا کلمات به بردار تبدیل شوند.

1. # Step 4: Download word embeddings and create an embedding matrix

2. def download\_and\_create\_embedding\_matrix(tokenizer):

3.

4. embedding\_file\_path = f'embeddings/wiki-news-300d-1M.vec'

5.

6. if not os.path.isfile(embedding\_file\_path):

7. print('Downloading word vectors ... ')

8. url = f'https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.vec.zip'

9. urllib.request.urlretrieve(url, f'wiki-news-300d-1M.vec.zip')

10.

11. print('Unzipping ... ')

12. with zipfile.ZipFile(f'wiki-news-300d-1M.vec.zip', 'r') as zip\_ref:

13. zip\_ref.extractall('embeddings')

14. print('done.')

15.

16. os.remove(f'wiki-news-300d-1M.vec.zip')

17.

18. embeddings\_index = {}

19. with open(embedding\_file\_path, encoding='utf-8') as f:

20. for line in f:

21. values = line.split()

22. word = values[0]

23. coefs = np.asarray(values[1:], dtype='float32')

24. embeddings\_index[word] = coefs

25.

26. embedding\_matrix = np.zeros((len(tokenizer.word\_index) + 1, 300))

27. for word, i in tokenizer.word\_index.items():

28. embedding\_vector = embeddings\_index.get(word)

29. if embedding\_vector is not None:

30. embedding\_matrix[i] = embedding\_vector

31.

32. return embedding\_matrix

مرحله 5 ساخت مدل دسته بند

مرحله 5-1-

در این مرحله مطابق اطلاعات داده شده مدل را آموزش دادیم و هیستوری را جهت نمایش نمودار بازگشت داده ایم:

1. # Step 5-1: Build and train a GRU model

2. def build\_and\_train\_gru\_model(embedding\_matrix, max\_length, input\_dim):

3. model = Sequential()

4. model.add(Embedding(input\_dim=input\_dim, output\_dim=embedding\_matrix.shape[1],

5. weights=[embedding\_matrix], input\_length=max\_length, trainable=False))

6. model.add(GRU(units=32, dropout=0.2, recurrent\_dropout=0.2))

7. model.add(Dense(1, activation='sigmoid'))

8.

9. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

10.

11. # Train the model

12. history = model.fit(padded\_train\_sequences, y\_train.iloc[selected\_samples\_train],

13. epochs=10, batch\_size=64, validation\_split=0.2)

14.

15. return model, history

خروجی کنسول این مرحله:

1. Epoch 1/10

2. 4/4 [==============================] - 10s 2s/step - loss: 0.6925 - accuracy: 0.5292 - val\_loss: 0.6911 - val\_accuracy: 0.5500

3. Epoch 2/10

4. 4/4 [==============================] - 6s 2s/step - loss: 0.6901 - accuracy: 0.5542 - val\_loss: 0.6897 - val\_accuracy: 0.5500

5. Epoch 3/10

6. 4/4 [==============================] - 6s 2s/step - loss: 0.6890 - accuracy: 0.5542 - val\_loss: 0.6887 - val\_accuracy: 0.5500

7. Epoch 4/10

8. 4/4 [==============================] - 7s 2s/step - loss: 0.6882 - accuracy: 0.5542 - val\_loss: 0.6882 - val\_accuracy: 0.5500

9. Epoch 5/10

10. 4/4 [==============================] - 10s 3s/step - loss: 0.6874 - accuracy: 0.5542 - val\_loss: 0.6882 - val\_accuracy: 0.5500

11. Epoch 6/10

12. 4/4 [==============================] - 9s 2s/step - loss: 0.6875 - accuracy: 0.5542 - val\_loss: 0.6883 - val\_accuracy: 0.5500

13. Epoch 7/10

14. 4/4 [==============================] - 9s 2s/step - loss: 0.6877 - accuracy: 0.5542 - val\_loss: 0.6884 - val\_accuracy: 0.5500

15. Epoch 8/10

16. 4/4 [==============================] - 9s 2s/step - loss: 0.6872 - accuracy: 0.5542 - val\_loss: 0.6883 - val\_accuracy: 0.5500

17. Epoch 9/10

18. 4/4 [==============================] - 7s 2s/step - loss: 0.6879 - accuracy: 0.5542 - val\_loss: 0.6882 - val\_accuracy: 0.5500

19. Epoch 10/10

20. 4/4 [==============================] - 7s 2s/step - loss: 0.6873 - accuracy: 0.5542 - val\_loss: 0.6882 - val\_accuracy: 0.5500

مرحله 2-5-

جهت نمایش نمودار از هیستوری مرحله قبل استفاده شده:

1. # Step 5-2: Plot loss and accuracy for the GRU model

2. def plot\_metrics(history, title):

3. plt.figure(figsize=(12, 4))

4.

5. # Plot training & validation accuracy values

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

7. plt.plot(history.history['accuracy'])

8. plt.plot(history.history['val\_accuracy'])

9. plt.title(title + ' - Model Accuracy')

10. plt.xlabel('Epoch')

11. plt.ylabel('Accuracy')

12. plt.legend(['Train', 'Validation'], loc='upper left')

13.

14. # Plot training & validation loss values

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

16. plt.plot(history.history['loss'])

17. plt.plot(history.history['val\_loss'])

18. plt.title(title + ' - Model Loss')

19. plt.xlabel('Epoch')

20. plt.ylabel('Loss')

21. plt.legend(['Train', 'Validation'], loc='upper left')

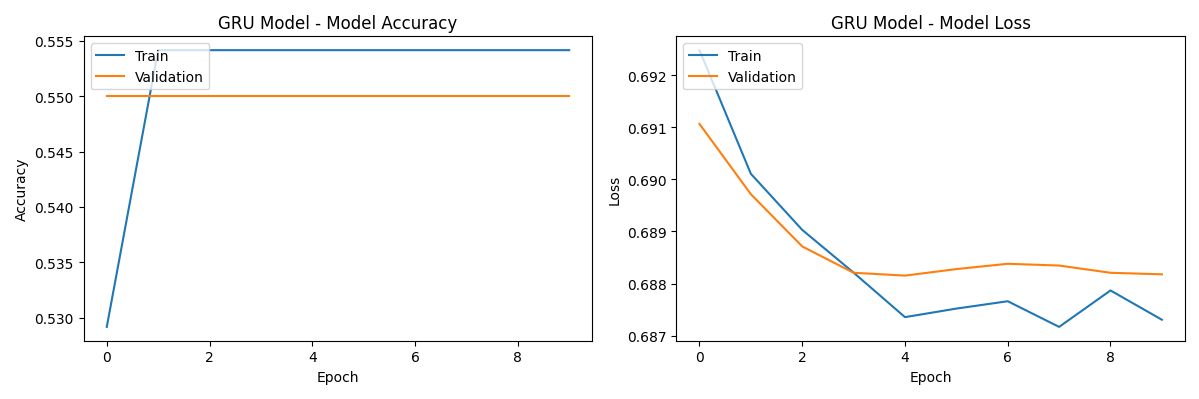
22.

23. plt.tight\_layout()

24. plt.show()

25.

خروجی:



مرحله 5-3-

در این مرحله نیز مطابق اطلاعات داده شده آموزش مدل انجام شده:

1. # Step 5-3: Build and train a BiGRU model

2. def build\_and\_train\_bigru\_model(embedding\_matrix, max\_length, input\_dim):

3. model = Sequential()

4. model.add(Embedding(input\_dim=input\_dim, output\_dim=embedding\_matrix.shape[1],

5. weights=[embedding\_matrix], input\_length=max\_length, trainable=False))

6. model.add(Bidirectional(GRU(units=32, dropout=0.2, recurrent\_dropout=0.2)))

7. model.add(Dense(1, activation='sigmoid'))

8.

9. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

10.

11. # Train the model

12. history = model.fit(padded\_train\_sequences, y\_train.iloc[selected\_samples\_train],

13. epochs=10, batch\_size=64, validation\_split=0.2)

14.

15. return model, history

خروجی کنسول این مرحله:

1. Epoch 1/10

2. 4/4 [==============================] - 30s 6s/step - loss: 0.6975 - accuracy: 0.4667 - val\_loss: 0.6871 - val\_accuracy: 0.5500

3. Epoch 2/10

4. 4/4 [==============================] - 22s 5s/step - loss: 0.6846 - accuracy: 0.5625 - val\_loss: 0.6867 - val\_accuracy: 0.5500

5. Epoch 3/10

6. 4/4 [==============================] - 20s 5s/step - loss: 0.6829 - accuracy: 0.5542 - val\_loss: 0.6868 - val\_accuracy: 0.5500

7. Epoch 4/10

8. 4/4 [==============================] - 23s 6s/step - loss: 0.6775 - accuracy: 0.5542 - val\_loss: 0.6856 - val\_accuracy: 0.5500

9. Epoch 5/10

10. 4/4 [==============================] - 23s 6s/step - loss: 0.6726 - accuracy: 0.5542 - val\_loss: 0.6835 - val\_accuracy: 0.5500

11. Epoch 6/10

12. 4/4 [==============================] - 22s 5s/step - loss: 0.6674 - accuracy: 0.5583 - val\_loss: 0.6820 - val\_accuracy: 0.5500

13. Epoch 7/10

14. 4/4 [==============================] - 23s 6s/step - loss: 0.6634 - accuracy: 0.5708 - val\_loss: 0.6808 - val\_accuracy: 0.5500

15. Epoch 8/10

16. 4/4 [==============================] - 22s 5s/step - loss: 0.6575 - accuracy: 0.5792 - val\_loss: 0.6797 - val\_accuracy: 0.5500

17. Epoch 9/10

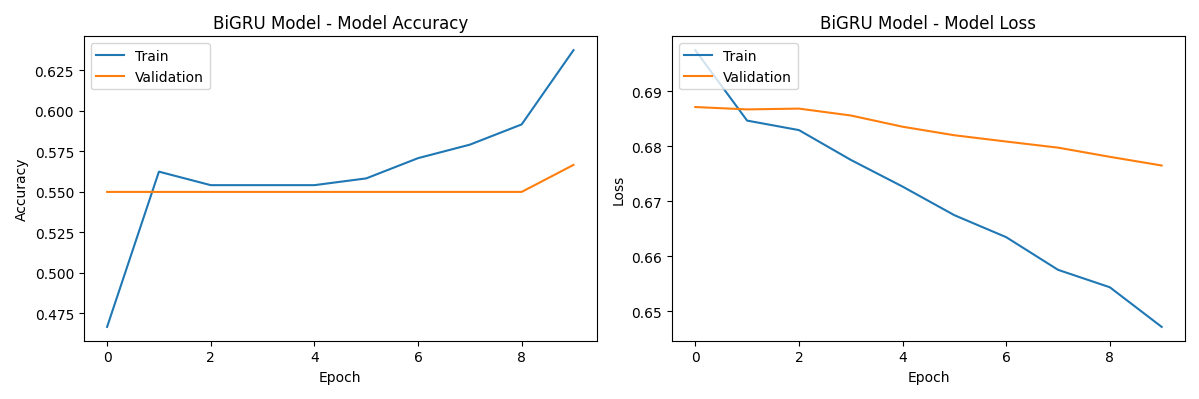
18. 4/4 [==============================] - 22s 6s/step - loss: 0.6543 - accuracy: 0.5917 - val\_loss: 0.6781 - val\_accuracy: 0.5500

19. Epoch 10/10

20. 4/4 [==============================] - 24s 6s/step - loss: 0.6471 - accuracy: 0.6375 - val\_loss: 0.6765 - val\_accuracy: 0.5667

مرحله 5-4-

در این مرحله با استفاده از تابع plot\_metrics **که در مرحله 5-3 نوشته شده بود خروجی نموداری گزارش شده:**

****

**در نهایت کد های مربوط به اجرای توابع به شرح زیر می باشد:**

1. file\_path = 'IMDB\_Dataset.csv'

2. df = load\_dataset(file\_path)

3.

4. # step1-1

5. convert\_labels(df)

6.

7. # step1-2

8. remove\_duplicates(df)

9.

10. # step1-3

11. df = random\_selection(df)

12.

13. # step1-4

14. df = get\_polarity(df)

15.

16. # step1-5

17. plot\_sentiment\_histograms(df)

18.

19. # step1-6

20. i = 0

21. for index, row in df.iterrows():

22. print(i)

23. i += 1

24. df.at[index, 'review'] = preprocess\_text(row['review'])

25.

26. df.to\_csv('preprocessed\_and\_split\_IMDB\_Dataset.csv', index=False)

27.

28. # step1-7

29.

30. file\_path = 'preprocessed\_and\_split\_IMDB\_Dataset.csv'

31. df = load\_dataset(file\_path)

32. X\_train, X\_test, y\_train, y\_test = split\_dataset(df)

33.

34. # # step2-1

35. bow\_matrix\_train, bow\_matrix\_test = create\_bow\_representation(X\_train, X\_test)

36.

37. # # Step 2-2

38. train\_clusters, test\_clusters = kmeans\_clustering(bow\_matrix\_train, bow\_matrix\_test)

39.

40. # # Step 2-3

41. visualize\_clusters(bow\_matrix\_train, train\_clusters, title='Training Set Clusters (t-SNE Visualization)')

42.

43. # # Step 2-4

44. num\_samples\_per\_cluster = 100 # Adjust the number of samples as needed

45. selected\_samples\_train = select\_uniform\_sample(bow\_matrix\_train, train\_clusters, num\_samples\_per\_cluster)

46. selected\_samples\_test = select\_uniform\_sample(bow\_matrix\_test, test\_clusters, num\_samples\_per\_cluster)

47.

48. # # Step 3-1

49. tokenizer, train\_sequences, test\_sequences = tokenize\_text\_sequences(X\_train.iloc[selected\_samples\_train], X\_test.iloc[selected\_samples\_test])

50.

51. # # Step 3-2

52. padded\_train\_sequences, padded\_test\_sequences, max\_length = pad\_text\_sequences(train\_sequences, test\_sequences)

53.

54. # step 4

55. embedding\_matrix = download\_and\_create\_embedding\_matrix(tokenizer)

56.

57. # Step 5-1 (GRU Model)

58. gru\_model, gru\_history = build\_and\_train\_gru\_model(embedding\_matrix, max\_length, input\_dim=len(tokenizer.word\_index) + 1)

59.

60. # Step 5-2 (Plot Metrics for GRU Model)

61. plot\_metrics(gru\_history, 'GRU Model')

62.

63. # Step 5-3 (BiGRU Model)

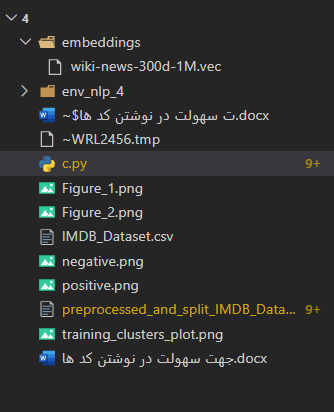
64. bigru\_model, bigru\_history = build\_and\_train\_bigru\_model(embedding\_matrix, max\_length, input\_dim=len(tokenizer.word\_index) + 1)

65.

66. # Step 5-4 (Plot Metrics for BiGRU Model)

67. plot\_metrics(bigru\_history, 'BiGRU Model')

**همچنین دایرکتوری به شرح زیر می باشد:**

****