Forum Discussions Categorization Report

CS_12

Name	ID
Narden Gerges Adward Amin	2021170576
Nada Abdelmoneem Ahmed	2021170583
Youssef Hany Ezzat Aly	2021170653
Mina Edwar Dawood Elias	2021170565
Dalia Abd Elazim Mohamed	2021170179
Yousef Emad El Din Ibrahim	2021170640

Without Preprocessing:

```
df=df.dropna()
category_mapping = {
    'Politics': 0,
    'Sports': 1,
    'Media': 2,
    'Market & Economy': 3,
    'STEM': 4
}
df['Discussion'] = df['Discussion'].astype(str)
df['Category'] = df['Category'].map(category_mapping)
df=df.drop('SampleID',axis=1)
```

With Preprocessing:

```
def preprocess_text(text):
    lemmatizer = WordNetLemmatizer()
    # Remove NaN values
    if pd.isnull(text):
        return ""
    # Lowercase the text
    text = text.lower()
    # Remove punctuation
    text = re.sub(r'[^a-zA-Z\s]', ' ', text)
    # Tokenize and remove stopwords
    words = word_tokenize(text)
    stop_words = set(stopwords.words('english'))

    words = [word for word in words if word not in stop_words]
    # Lemmatization
    words = [lemmatizer.lemmatize(word) for word in words]
    # Join back into a single string
    return ' '.join(words)

# Apply preprocessing to the 'Discussion' column
df['Discussion']=df['Discussion'].astype(str)
df['Discussion'] = df['Discussion'].dropna().apply(preprocess_text)
```

Augmentation

We tested it and observed that it had no significant impact on accuracy.

```
from nltk.corpus import wordnet

def synonym_replacement(row):
    text = row['Discussion']
    words = text.split()
    augmented_text = []
    for word in words:
        # Get synonyms for each word
        synonyms = wordnet.synsets(word)
        if synonyms:
            # Replace the word with a synonym
            synonym = synonyms[0].lemmas()[0].name()
            augmented_text.append(synonym)
        else:
            augmented_text.append(word)
    return {'augmented': augmented_text, 'category': row['Category']}

random_sample = df.sample(n=5000, random_state=42)
augmented_data = random_sample.apply(synonym_replacement, axis=1)
augmented_df = pd.DataFrame(augmented_data.tolist())
augmented_df['augmented'] = augmented_df['augmented'].apply(lambda x: " ".join(x))
augmented_df.rename(columns={'augmented': 'Discussion', 'category': 'Category'}, inplace=True)
df = pd.concat([df, augmented_df], ignore_index=True)
```

Tokenizer

This code tokenizes text data using Keras' **`Tokenizer`**, converts text into padded sequences with a fixed length of 100, and handles out-of-vocabulary words using a placeholder (<OOV>). After testing various configurations, we found **`maxlen=100`** and **`vocab_size=30000`** to yield the best results.

```
#####Tokenization########
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

vocab_size = 30000
oov_token = "<OOV>"
padding_type = "post"
trunc_type = "post"
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)
tokenizer.fit_on_texts(df['Discussion'])
training_sequences = tokenizer.texts_to_sequences(df['Discussion'])
training_padded = pad_sequences(training_sequences, maxlen=100, padding=padding_type, truncating=trunc_t
X=np.array(training_padded)
labels = np.array(df['Category'])
```

Splitting

This code splits the dataset into training and validation sets (**`x_train`**, **`x_val`**, **`y_train`**, **`y_val`**) with an 80-20 ratio, using **`stratify=labels`** to ensure class distribution is maintained. Stratification improves model accuracy by preventing class imbalance in the splits.

```
from sklearn.model_selection import train_test_split

x_train, x_val, y_train, y_val = train_test_split(X, labels, test_size=0.2, random_state=42,stratify=lab
```

Word embedding

This code loads GloVe word embeddings from a specified file and stores them in a dictionary. It then initializes an embedding matrix of size（**`vocab_size, embedding_dim`**）and populates it with vectors corresponding to words in the tokenizer's vocabulary. Words not found in GloVe are represented by zero vectors.

```
# # Function to load GloVe vectors
def load_glove_vectors(filepath):
    embeddings_index = {}
    with open(filepath, encoding="utf8") as f:
        for line in f:
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], dtype="float32")
            embeddings_index[word] = vector
    return embeddings_index

# Load GloVe embeddings
glove_path = "/content/drive/MyDrive/Glove/glove.6B.300d.txt"
glove_vectors = load_glove_vectors(glove_path)
# Parameters
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 300

# Initialize embedding matrix
embedding_matrix = np.zeros((vocab_size, embedding_dim))

# Populate embedding matrix
for word, i in tokenizer.word_index.items():
    if word in glove_vectors:
        embedding_matrix[i] = glove_vectors[word]
```

Normal Models

BILSTM

Architecture

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 300)	14,587,200
bidirectional (Bidirectional)	(None, 100, 512)	1,140,736
global_average_pooling1d (GlobalAveragePooling1D)	(None, 512)	0
dense (Dense)	(None, 128)	65,664
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 5)	325

Total params: 15,802,181 (60.28 MB)

Trainable params: 1,214,981 (4.63 MB)

Non-trainable params: 14,587,200 (55.65 MB)

Accuracy

- Model Accuracy (training and validation) with preprocessing:

```
Epoch 7/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 146s 930ms/step - accuracy: 0.7354 - loss: 0.6975 - val_accuracy: 0.7281 - val_loss: 0.7273
Epoch 8/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 138s 880ms/step - accuracy: 0.7449 - loss: 0.6710 - val_accuracy: 0.7315 - val_loss: 0.7241
Epoch 9/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 140s 891ms/step - accuracy: 0.7500 - loss: 0.6554 - val_accuracy: 0.7261 - val_loss: 0.7333
Epoch 10/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 139s 888ms/step - accuracy: 0.7600 - loss: 0.6295 - val_accuracy: 0.7331 - val_loss: 0.7332
```

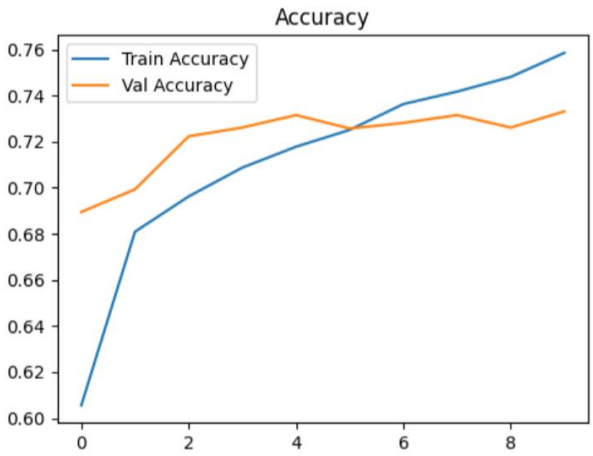
- Model Accuracy (training and validation) without preprocessing:

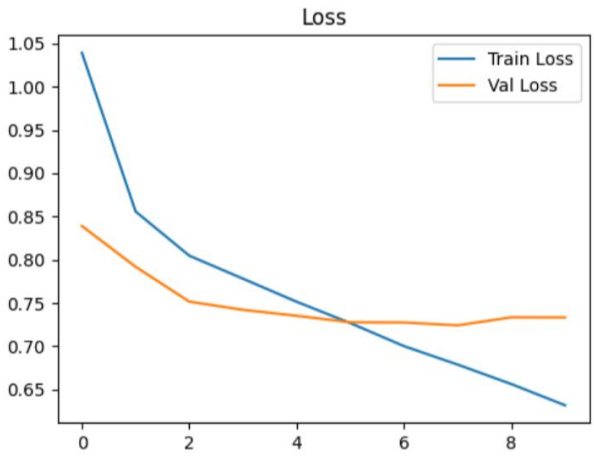
```
Epoch 8/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 82s 320ms/step - accuracy: 0.7432 - loss: 0.6808 - val_accuracy: 0.7231 - val_loss: 0.7339
Epoch 9/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 48s 311ms/step - accuracy: 0.7475 - loss: 0.6713 - val_accuracy: 0.7274 - val_loss: 0.7227
Epoch 10/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 48s 310ms/step - accuracy: 0.7624 - loss: 0.6347 - val_accuracy: 0.7278 - val_loss: 0.7559
```

Model evaluation: (74%)

```
bilstmModel.evaluate(x_val,y_val)
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 20s 128ms/step - accuracy: 0.7407 - loss: 0.7111
[0.7240527272224426, 0.7314925789833069]
```





GRU

Architecture

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 100, 300)	14,587,200
gru (GRU)	(None, 100, 256)	428,544
dropout_2 (Dropout)	(None, 100, 256)	0
global_max_pooling1d (GlobalMaxPooling1D)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32,896
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8,256
dropout_4 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 5)	325

Total params: 15,057,221 (57.44 MB)
Trainable params: 470,021 (1.79 MB)
Non-trainable params: 14,587,200 (55.65 MB)

- Model Accuracy (training and validation) without preprocessing:

```
Epoch 6/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - accuracy: 0.7763 - loss: 0.6125 - val_accuracy: 0.7124 - val_loss: 0.7615
Epoch 7/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.7848 - loss: 0.5806 - val_accuracy: 0.7108 - val_loss: 0.7721
Epoch 8/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - accuracy: 0.7883 - loss: 0.5752 - val accuracy: 0.7231 - val loss: 0.7593
```

- Model Accuracy (training and validation) with preprocessing:

```
Epoch 5/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.7554 - loss: 0.6539 - val_accuracy: 0.7241 - val_loss: 0.7459
Epoch 6/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.7729 - loss: 0.6081 - val_accuracy: 0.7241 - val_loss: 0.7353
Epoch 7/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.7968 - loss: 0.5496 - val_accuracy: 0.7037 - val_loss: 0.7770
```

Model evaluation

```
✓ 1s [62] gruModel.evaluate(x_val,y_val)

155/155 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7230 - loss: 0.7459
[0.7402076721191406, 0.7210953235626221]
```

Transformer

Architecture

Layer (type)	Output Shape	Param #	Connected to
input_layer_17 (InputLayer)	(None, 100)	0	-
embedding_14 (Embedding)	(None, 100, 300)	18,212,700	input_layer_17[0][0]
multi_head_attention_18 (MultiHeadAttention)	(None, 100, 300)	308,268	embedding_14[0][0], embedding_14[0][0]
add_12 (Add)	(None, 100, 300)	0	multi_head_attention_…, multi_head_attention_…
layer_normalization_26 (LayerNormalization)	(None, 100, 300)	600	add_12[0][0]
flatten_4 (Flatten)	(None, 30000)	0	layer_normalization_2…
dense_28 (Dense)	(None, 64)	1,920,064	flatten_4[0][0]
dropout_27 (Dropout)	(None, 64)	0	dense_28[0][0]
dense_29 (Dense)	(None, 5)	325	dropout_27[0][0]

Total params: 20,441,957 (77.98 MB)
Trainable params: 2,229,257 (8.50 MB)
Non-trainable params: 18,212,700 (69.48 MB)

Accuracy

- Model Accuracy (training and validation) without preprocessing:

```
Epoch 8/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.6971 - loss: 0.7983 - val_accuracy: 0.7006 - val_loss: 0.8325
Epoch 9/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.6930 - loss: 0.8105 - val_accuracy: 0.7053 - val_loss: 0.8219
Epoch 10/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.7001 - loss: 0.7692 - val_accuracy: 0.6890 - val_loss: 0.8353
```

Bert

```
Total params: 108,600,581 (414.28 MB)
Trainable params: 108,600,581 (414.28 MB)
Non-trainable params: 0 (0.00 B)
```

Roberta

```
Total params: 124,253,957 (473.99 MB)
Trainable params: 124,253,957 (473.99 MB)
Non-trainable params: 0 (0.00 B)
```

Ensemble

Other trials

RoBERTa + DistilRoBERTa

1. We tried removing the alpha, with max Length=128,Learning rate 0.00003,batch size=32, weight decay= 0.01 and weights= [1,0.6] with Accuracy : 77.1%

2. We tried, with max Length=512 ,Learning rate 0.00002,batch size=32, weight decay= 0.01 and weights= [1,0.7] with Accuracy : 77.6%

3. We tried removing the alpha, with max Length=128,Learning rate 0.00003,batch size=16, weight decay= 0.01 and weights= [1,0.6] with Accuracy : 77.1%

RoBERTa + BERT

1. We tried removing the alpha, with max Length=128,Learning rate 0.00003,batch size=32, weight decay= 0.01 and weights= [1,0.6] with Accuracy : 78.2%

2. We tried, with max Length=512 ,Learning rate 0.00002,batch size=32, weight decay= 0.01 and weights= [1,0.7] with Accuracy : 78.6%

3. We tried removing the alpha, with max Length=128,Learning rate 0.00003,batch size=16, weight decay= 0.01 and weights= [1,0.6] with Accuracy : 78.1%

Conclusion

Finally, what got us the highest accuracy was : Learning rate 0.00002, Weights[1,0.5] for RoBERTa and BERT respectively, with 4 epochs and batch size = 32,Max length =128