Comparing the Classification Performance of Various Models on News Headlines

Aditya Rathod (AGR190000)

April 22, 2023

Kaggle dataset

This is a dataset of news headlines and their categories. In this problem, we will tokenize the dataset on each word and feed it as sequence data into various models.

The dataset is not split and is in JSON format, requiring heavy preprocessing to be loaded by Keras. I also use the `tf.data` API to load data in a RAM-efficient manner.

1 Preprocessing

1.1 Load data

[ ]:
```
import pandas as pd
import tensorflow as tf
```

[ ]:
```
df = pd.read_json(
    './data/News_Category_Dataset_v3.json',
    dtype={'category': 'category'},
    lines=True
)
df = df[['headline', 'category']]
```

1.2 Preprocess data

Save to file for Keras text dataset handling:

[ ]:
```
import os
import re
from pathlib import Path
from sklearn.model_selection import train_test_split

def remove_special_characters(input_string):
    return re.sub('[^A-Za-z0-9]+', '', input_string)

def save_row_subset(subset):
    if subset != 'train' and subset != 'test':
        raise ValueError('Must be one of {train, test}')
    return lambda row: save_row(row, subset)
```

```
def save_row(row, subset):
    label = row['category']
    headline = row['headline']
    idx = row.name
    save_path_root = Path(f'./data/ByCategory/{subset}/
 ↪{remove_special_characters(label)}').resolve()
    if not os.path.isdir(save_path_root):
        os.makedirs(save_path_root)
    save_path = save_path_root / f'{idx}.txt'

    with open(save_path, 'w') as f:
        f.write(headline + "\n")


train_df, test_df = train_test_split(df, test_size=0.20, random_state=42)

train_df.apply(save_row_subset('train'), axis=1)
test_df.apply(save_row_subset('test'), axis=1)
```

[ ]:

```
128310    None
139983    None
42339     None
131494    None
163649    None
          …
91721     None
10964     None
140604    None
182108    None
28078     None
Length: 41906, dtype: object
```

Create Keras datasets:

[ ]:

```
BATCH_SIZE = 128
seed = 42

# load into TF dataset
raw_train_ds, raw_val_ds = tf.keras.utils.text_dataset_from_directory(
    './data/ByCategory/train',
    batch_size=BATCH_SIZE,
    label_mode='categorical',
    validation_split=0.1,
    subset='both',
    seed=seed
)
```

```
raw_test_ds = tf.keras.utils.text_dataset_from_directory(
    './data/ByCategory/test',
    batch_size=BATCH_SIZE,
    label_mode='categorical',
    seed=seed
)
```

```
Found 167621 files belonging to 42 classes.
Using 150859 files for training.
Using 16762 files for validation.
Found 41906 files belonging to 42 classes.
```

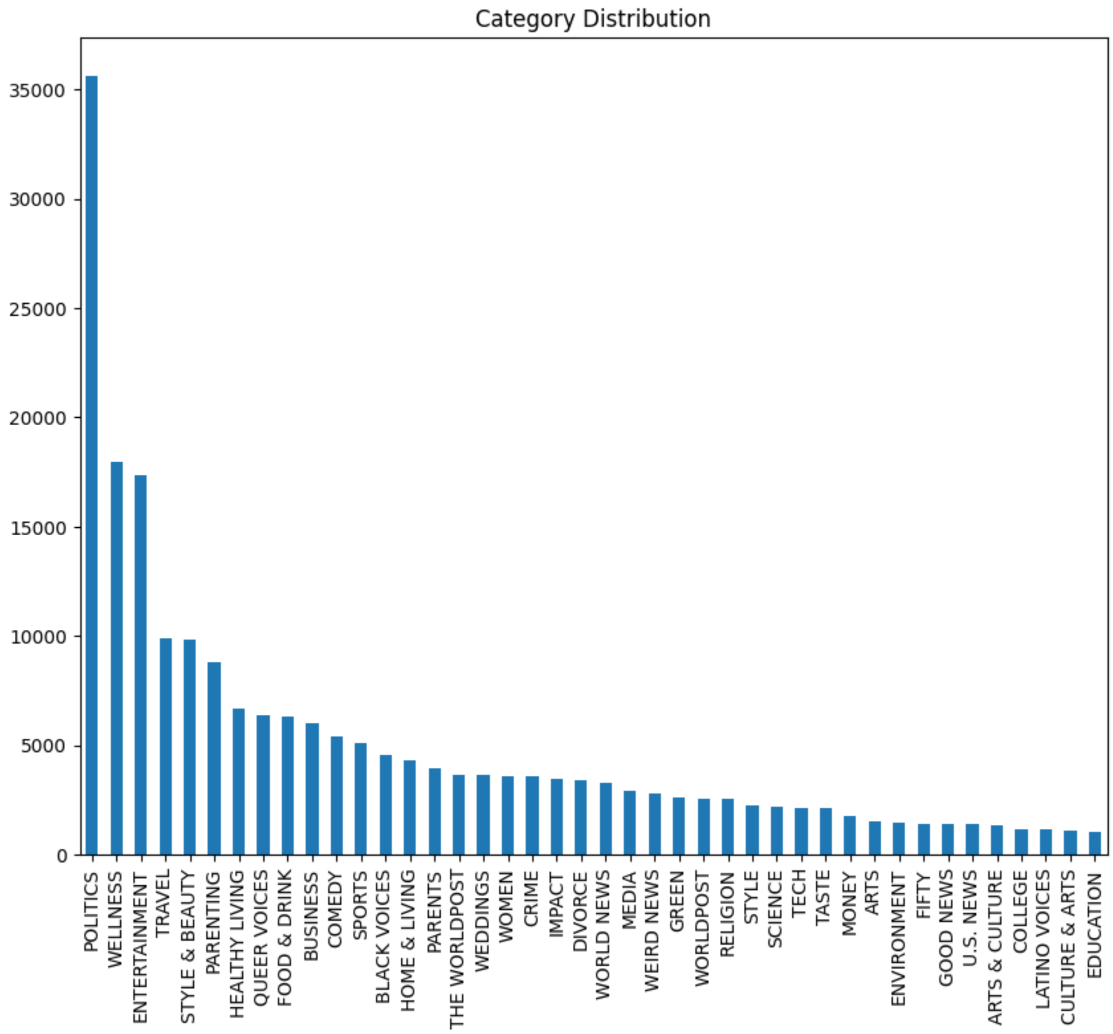
Vectorize data:

[ ]:
```
# vectorize sequences to length 120
VOCAB_SIZE = 200000
MAX_SEQUENCE_LENGTH = 120

vectorize_layer = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    output_sequence_length=MAX_SEQUENCE_LENGTH
)

train_text = raw_train_ds.map(lambda text, labels: text)
val_text = raw_val_ds.map(lambda text, labels: text)
vectorize_layer.adapt(train_text)
vectorize_layer.adapt(val_text)

def vectorize_text(text, label):
    text = tf.expand_dims(text, -1)
    return vectorize_layer(text), label

AUTOTUNE = tf.data.AUTOTUNE

def configure_dataset(dataset):
    return dataset.cache().prefetch(buffer_size=AUTOTUNE)

train_ds = configure_dataset(raw_train_ds.map(vectorize_text))
val_ds = configure_dataset(raw_val_ds.map(vectorize_text))
test_ds = configure_dataset(raw_test_ds.map(vectorize_text))
```

1.3 Visualize class distribution

[ ]:
```
df['category'].value_counts().plot.bar(figsize=(10,8), title="Category␣
↪Distribution")
```

[ ]:
```
<Axes: title={'center': 'Category Distribution'}>
```



We can see that most of the articles in the dataset are politics, followed by wellness, entertainment, travel, then style and beauty. However, there are disproportionately more politics articles, which may affect classification accuracy.

The model should be able to predict the category of a new article given its headline. This can be used in news aggregator websites to determine under which section to put an article under.

2 Train basic sequential model

2.1 Define helper function to create models

We will use this to create models in a quick and easy manner, abstracting away the training and evaluation steps in a standardized manner.

[ ]:

```
```

```
# helper function to create models
def eval_model(middle_layers, model_name='model', embedding_size=128, has_lstm=False, lstm_size=30, num_epochs=10, random_seed=42, return_model=False):
    tf.keras.utils.set_random_seed(random_seed)
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Input(shape=(120,)))
    model.add(tf.keras.layers.Embedding(VOCAB_SIZE + 1, embedding_size, mask_zero=True))
    if not has_lstm:
        model.add(tf.keras.layers.Flatten())
    else:
        model.add(tf.keras.layers.LSTM(lstm_size))
    if middle_layers is not None:
        for layer in middle_layers:
            model.add(layer)
    model.add(tf.keras.layers.Dense(42, activation='relu'))
    model.compile(
        loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
        optimizer='adam',
        metrics=['accuracy', tf.keras.metrics.Precision(name='precision'), tf.keras.metrics.Recall(name='recall')]
    )
    model.summary()
    history = model.fit(
        train_ds,
        validation_data=val_ds,
        epochs=num_epochs
    )
    test_loss, test_acc, test_precision, test_recall = model.evaluate(test_ds)
    train_loss, train_acc, train_precision, train_recall = history.history['loss'][-1], history.history['accuracy'][-1], history.history['precision'][-1], history.history['recall'][-1]
    val_loss, val_acc, val_precision, val_recall = history.history['val_loss'][-1], history.history['val_accuracy'][-1], history.history['val_precision'][-1], history.history['val_recall'][-1]

    metrics = {
        'model_name': model_name,
        'train_loss': train_loss,
        'train_acc': train_acc,
        'train_precision': train_precision,
        'train_recall': train_recall,
        'train_f1': 2 * (train_precision * train_recall) / (train_precision + train_recall),
        'test_loss': test_loss,
```

```
        'test_acc': test_acc,
        'test_precision': test_precision,
        'test_recall': test_recall,
        'test_f1': 2 * (test_precision * test_recall) / (test_precision +␣
↪test_recall),
        'val_loss': val_loss,
        'val_acc': val_acc,
        'val_precision': test_precision,
        'val_recall': test_recall,
        'val_f1': 2 * (val_precision * val_recall) / (val_precision +␣
↪val_recall),
    }

    if return_model:
        return model, metrics
    return metrics
```

3 Basic model creation

This simple sequential model has a 128-dimensional embedding with a flattening layer, 2 dense layers with 100 units each, sandwiched between a 20% dropout layer, and an output layer of 42 units.

[ ]:
```
seq_1_model, seq_1_metrics = eval_model([
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(100, activation='relu')
], return_model=True, model_name='sequential_model_basic')
```

```
Model: "sequential_8"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_8 (Embedding)     (None, 120, 128)          25600128

 flatten_8 (Flatten)         (None, 15360)             0

 dense_24 (Dense)            (None, 100)               1536100

 dropout_8 (Dropout)         (None, 100)               0

 dense_25 (Dense)            (None, 100)               10100

 dense_26 (Dense)            (None, 42)                4242

=================================================================
Total params: 27,150,570
```

```
Trainable params: 27,150,570
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1179/1179 [==============================] - 33s 26ms/step - loss: 2.7352 -
accuracy: 0.3439 - precision: 0.0988 - recall: 0.5432 - val_loss: 2.3931 -
val_accuracy: 0.4316 - val_precision: 0.1199 - val_recall: 0.5773
Epoch 2/10
1179/1179 [==============================] - 13s 11ms/step - loss: 2.2929 -
accuracy: 0.4496 - precision: 0.1369 - recall: 0.5927 - val_loss: 2.2835 -
val_accuracy: 0.4544 - val_precision: 0.1507 - val_recall: 0.5910
Epoch 3/10
1179/1179 [==============================] - 12s 10ms/step - loss: 2.0629 -
accuracy: 0.4944 - precision: 0.1712 - recall: 0.6074 - val_loss: 2.3459 -
val_accuracy: 0.4494 - val_precision: 0.1752 - val_recall: 0.5801
Epoch 4/10
1179/1179 [==============================] - 12s 10ms/step - loss: 1.8743 -
accuracy: 0.5374 - precision: 0.2024 - recall: 0.6198 - val_loss: 2.4445 -
val_accuracy: 0.4521 - val_precision: 0.1682 - val_recall: 0.6037
Epoch 5/10
1179/1179 [==============================] - 12s 10ms/step - loss: 1.6896 -
accuracy: 0.5883 - precision: 0.2159 - recall: 0.6550 - val_loss: 2.5694 -
val_accuracy: 0.4464 - val_precision: 0.1988 - val_recall: 0.5830
Epoch 6/10
1179/1179 [==============================] - 12s 10ms/step - loss: 1.5717 -
accuracy: 0.6132 - precision: 0.2576 - recall: 0.6572 - val_loss: 2.7165 -
val_accuracy: 0.4392 - val_precision: 0.2307 - val_recall: 0.5609
Epoch 7/10
1179/1179 [==============================] - 12s 10ms/step - loss: 1.4980 -
accuracy: 0.6281 - precision: 0.2898 - recall: 0.6587 - val_loss: 2.8922 -
val_accuracy: 0.4320 - val_precision: 0.2541 - val_recall: 0.5437
Epoch 8/10
1179/1179 [==============================] - 13s 11ms/step - loss: 1.4494 -
accuracy: 0.6369 - precision: 0.3226 - recall: 0.6592 - val_loss: 3.0063 -
val_accuracy: 0.4311 - val_precision: 0.2562 - val_recall: 0.5419
Epoch 9/10
1179/1179 [==============================] - 12s 10ms/step - loss: 1.4154 -
accuracy: 0.6434 - precision: 0.3410 - recall: 0.6601 - val_loss: 3.1700 -
val_accuracy: 0.4326 - val_precision: 0.2721 - val_recall: 0.5345
Epoch 10/10
1179/1179 [==============================] - 12s 10ms/step - loss: 1.3923 -
accuracy: 0.6467 - precision: 0.3570 - recall: 0.6605 - val_loss: 3.3365 -
val_accuracy: 0.4337 - val_precision: 0.2659 - val_recall: 0.5410
328/328 [==============================] - 1s 4ms/step - loss: 3.3581 -
accuracy: 0.4153 - precision: 0.2641 - recall: 0.5262
```

4 Train LSTM model

This simple LSTM model has a 128-dimensional embedding, a 30-unit LSTM layer, and a 100-unit dense layer with an output layer of 42 units.

[31]:
```
lstm_1_model, lstm_1_metrics = eval_model([
    tf.keras.layers.Dense(100, activation='relu'),
], has_lstm=True, return_model=True, model_name='lstm_model_basic')
```

```
Model: "sequential_10"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 120, 128)          25600128

 lstm_1 (LSTM)               (None, 30)                19080

 dense_29 (Dense)            (None, 100)               3100

 dense_30 (Dense)            (None, 42)                4242

=================================================================
Total params: 25,626,550
Trainable params: 25,626,550
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1179/1179 [==============================] - 55s 33ms/step - loss: 2.4751 -
accuracy: 0.3962 - precision: 0.0873 - recall: 0.6701 - val_loss: 2.0761 -
val_accuracy: 0.4950 - val_precision: 0.1074 - val_recall: 0.7286
Epoch 2/10
1179/1179 [==============================] - 18s 15ms/step - loss: 1.9659 -
accuracy: 0.5212 - precision: 0.1157 - recall: 0.7334 - val_loss: 1.9814 -
val_accuracy: 0.5245 - val_precision: 0.1253 - val_recall: 0.7251
Epoch 3/10
1179/1179 [==============================] - 19s 16ms/step - loss: 1.8033 -
accuracy: 0.5611 - precision: 0.1356 - recall: 0.7386 - val_loss: 1.9870 -
val_accuracy: 0.5259 - val_precision: 0.1440 - val_recall: 0.7151
Epoch 4/10
1179/1179 [==============================] - 19s 16ms/step - loss: 1.6834 -
accuracy: 0.5925 - precision: 0.1559 - recall: 0.7406 - val_loss: 2.0397 -
val_accuracy: 0.5219 - val_precision: 0.1612 - val_recall: 0.7062
Epoch 5/10
1179/1179 [==============================] - 19s 16ms/step - loss: 1.5796 -
accuracy: 0.6184 - precision: 0.1784 - recall: 0.7414 - val_loss: 2.1323 -
val_accuracy: 0.5144 - val_precision: 0.1775 - val_recall: 0.6914
Epoch 6/10
1179/1179 [==============================] - 19s 16ms/step - loss: 1.4873 -
```

```
accuracy: 0.6406 - precision: 0.2021 - recall: 0.7417 - val_loss: 2.2687 -
val_accuracy: 0.5056 - val_precision: 0.1922 - val_recall: 0.6765
Epoch 7/10
1179/1179 [==============================] - 18s 15ms/step - loss: 1.4066 -
accuracy: 0.6598 - precision: 0.2263 - recall: 0.7418 - val_loss: 2.4104 -
val_accuracy: 0.4973 - val_precision: 0.2068 - val_recall: 0.6616
Epoch 8/10
1179/1179 [==============================] - 19s 16ms/step - loss: 1.3393 -
accuracy: 0.6741 - precision: 0.2497 - recall: 0.7417 - val_loss: 2.5622 -
val_accuracy: 0.4901 - val_precision: 0.2176 - val_recall: 0.6510
Epoch 9/10
1179/1179 [==============================] - 18s 15ms/step - loss: 1.2797 -
accuracy: 0.6867 - precision: 0.2717 - recall: 0.7418 - val_loss: 2.7240 -
val_accuracy: 0.4783 - val_precision: 0.2312 - val_recall: 0.6349
Epoch 10/10
1179/1179 [==============================] - 19s 16ms/step - loss: 1.2306 -
accuracy: 0.6979 - precision: 0.2935 - recall: 0.7418 - val_loss: 2.8824 -
val_accuracy: 0.4775 - val_precision: 0.2424 - val_recall: 0.6273
328/328 [==============================] - 2s 5ms/step - loss: 2.7918 -
accuracy: 0.4723 - precision: 0.2441 - recall: 0.6262
```

5 Create larger embeddings for both models

These models are the same as the ones above, just with larger embeddings.

[32]:
```
seq_2_model, seq_2_metrics = eval_model([
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(100, activation='relu')
], return_model=True, embedding_size=256,␣
 ↪model_name='sequential_model_basic_big_embedding')
```

```
Model: "sequential_11"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, 120, 256)          51200256

 flatten_9 (Flatten)         (None, 30720)             0

 dense_31 (Dense)            (None, 100)               3072100

 dropout_9 (Dropout)         (None, 100)               0

 dense_32 (Dense)            (None, 100)               10100

 dense_33 (Dense)            (None, 42)                4242
```

```
=================================================================
Total params: 54,286,698
Trainable params: 54,286,698
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1179/1179 [==============================] - 55s 45ms/step - loss: 2.9021 -
accuracy: 0.3029 - precision: 0.0986 - recall: 0.4648 - val_loss: 2.6085 -
val_accuracy: 0.3767 - val_precision: 0.1254 - val_recall: 0.4976
Epoch 2/10
1179/1179 [==============================] - 20s 17ms/step - loss: 2.3564 -
accuracy: 0.4419 - precision: 0.1212 - recall: 0.6027 - val_loss: 2.2367 -
val_accuracy: 0.4729 - val_precision: 0.1235 - val_recall: 0.6426
Epoch 3/10
1179/1179 [==============================] - 20s 17ms/step - loss: 1.9956 -
accuracy: 0.5223 - precision: 0.1452 - recall: 0.6584 - val_loss: 2.2717 -
val_accuracy: 0.4730 - val_precision: 0.1497 - val_recall: 0.6259
Epoch 4/10
1179/1179 [==============================] - 22s 19ms/step - loss: 1.7714 -
accuracy: 0.5709 - precision: 0.1780 - recall: 0.6662 - val_loss: 2.3965 -
val_accuracy: 0.4598 - val_precision: 0.1800 - val_recall: 0.5950
Epoch 5/10
1179/1179 [==============================] - 22s 19ms/step - loss: 1.6068 -
accuracy: 0.6048 - precision: 0.2143 - recall: 0.6694 - val_loss: 2.5573 -
val_accuracy: 0.4537 - val_precision: 0.2075 - val_recall: 0.5766
Epoch 6/10
1179/1179 [==============================] - 22s 19ms/step - loss: 1.4950 -
accuracy: 0.6285 - precision: 0.2461 - recall: 0.6721 - val_loss: 2.7860 -
val_accuracy: 0.4412 - val_precision: 0.2252 - val_recall: 0.5548
Epoch 7/10
1179/1179 [==============================] - 20s 17ms/step - loss: 1.4266 -
accuracy: 0.6423 - precision: 0.2689 - recall: 0.6735 - val_loss: 2.9067 -
val_accuracy: 0.4360 - val_precision: 0.2267 - val_recall: 0.5511
Epoch 8/10
1179/1179 [==============================] - 20s 17ms/step - loss: 1.3842 -
accuracy: 0.6509 - precision: 0.2849 - recall: 0.6745 - val_loss: 2.9658 -
val_accuracy: 0.4372 - val_precision: 0.2396 - val_recall: 0.5440
Epoch 9/10
1179/1179 [==============================] - 19s 16ms/step - loss: 1.3504 -
accuracy: 0.6585 - precision: 0.3004 - recall: 0.6752 - val_loss: 3.1576 -
val_accuracy: 0.4281 - val_precision: 0.2296 - val_recall: 0.5419
Epoch 10/10
1179/1179 [==============================] - 21s 18ms/step - loss: 1.3326 -
accuracy: 0.6618 - precision: 0.3059 - recall: 0.6757 - val_loss: 3.2077 -
val_accuracy: 0.4303 - val_precision: 0.2589 - val_recall: 0.5277
328/328 [==============================] - 1s 3ms/step - loss: 3.2146 -
accuracy: 0.4160 - precision: 0.2584 - recall: 0.5151
```

[34]:
```
lstm_2_model, lstm_2_metrics = eval_model([
    tf.keras.layers.Dense(100, activation='relu'),
], has_lstm=True, embedding_size=256, return_model=True,␣
 ↪model_name='lstm_model_basic_big_embedding')
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_13 (Embedding)    (None, 120, 256)          51200256

 lstm_3 (LSTM)               (None, 30)                34440

 dense_36 (Dense)            (None, 100)               3100

 dense_37 (Dense)            (None, 42)                4242

=================================================================
Total params: 51,242,038
Trainable params: 51,242,038
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1179/1179 [==============================] - 51s 38ms/step - loss: 2.4599 -
accuracy: 0.4175 - precision: 0.0886 - recall: 0.6511 - val_loss: 2.0810 -
val_accuracy: 0.5159 - val_precision: 0.1096 - val_recall: 0.7060
Epoch 2/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.9609 -
accuracy: 0.5392 - precision: 0.1217 - recall: 0.7121 - val_loss: 2.0121 -
val_accuracy: 0.5321 - val_precision: 0.1321 - val_recall: 0.7013
Epoch 3/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.7882 -
accuracy: 0.5805 - precision: 0.1458 - recall: 0.7183 - val_loss: 2.0570 -
val_accuracy: 0.5273 - val_precision: 0.1522 - val_recall: 0.6880
Epoch 4/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.6572 -
accuracy: 0.6111 - precision: 0.1715 - recall: 0.7205 - val_loss: 2.1547 -
val_accuracy: 0.5163 - val_precision: 0.1702 - val_recall: 0.6708
Epoch 5/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.5434 -
accuracy: 0.6372 - precision: 0.1989 - recall: 0.7213 - val_loss: 2.2838 -
val_accuracy: 0.5067 - val_precision: 0.1911 - val_recall: 0.6486
Epoch 6/10
1179/1179 [==============================] - 25s 21ms/step - loss: 1.4466 -
accuracy: 0.6576 - precision: 0.2286 - recall: 0.7218 - val_loss: 2.4252 -
val_accuracy: 0.4976 - val_precision: 0.2135 - val_recall: 0.6314
Epoch 7/10
1179/1179 [==============================] - 25s 21ms/step - loss: 1.3687 -
```

```
accuracy: 0.6733 - precision: 0.2580 - recall: 0.7220 - val_loss: 2.5856 -
val_accuracy: 0.4916 - val_precision: 0.2309 - val_recall: 0.6163
Epoch 8/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.3072 -
accuracy: 0.6851 - precision: 0.2864 - recall: 0.7218 - val_loss: 2.7519 -
val_accuracy: 0.4871 - val_precision: 0.2407 - val_recall: 0.6059
Epoch 9/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.2578 -
accuracy: 0.6940 - precision: 0.3129 - recall: 0.7218 - val_loss: 2.8777 -
val_accuracy: 0.4783 - val_precision: 0.2530 - val_recall: 0.5934
Epoch 10/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.2194 -
accuracy: 0.7006 - precision: 0.3365 - recall: 0.7220 - val_loss: 3.0744 -
val_accuracy: 0.4744 - val_precision: 0.2608 - val_recall: 0.5864
328/328 [==============================] - 2s 6ms/step - loss: 2.9822 -
accuracy: 0.4708 - precision: 0.2657 - recall: 0.5882
```

6 Bigger LSTM model

LSTM model with larger embeddings, more LSTM units, and similar dense layer config as the sequential one.

[35]:
```
lstm_3_model, lstm_3_metrics = eval_model([
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(100, activation='relu')
], has_lstm=True, lstm_size=100, embedding_size=256, return_model=True,␣
 ↪model_name='lstm_model_large')
```

```
Model: "sequential_14"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_14 (Embedding)    (None, 120, 256)          51200256

 lstm_4 (LSTM)               (None, 100)               142800

 dense_38 (Dense)            (None, 100)               10100

 dropout_10 (Dropout)        (None, 100)               0

 dense_39 (Dense)            (None, 100)               10100

 dense_40 (Dense)            (None, 42)                4242

=================================================================
Total params: 51,367,498
Trainable params: 51,367,498
```

```
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1179/1179 [==============================] - 53s 39ms/step - loss: 2.6065 -
accuracy: 0.3917 - precision: 0.1144 - recall: 0.5610 - val_loss: 2.2677 -
val_accuracy: 0.4860 - val_precision: 0.1470 - val_recall: 0.6040
Epoch 2/10
1179/1179 [==============================] - 27s 23ms/step - loss: 2.2061 -
accuracy: 0.4987 - precision: 0.1566 - recall: 0.6050 - val_loss: 2.2191 -
val_accuracy: 0.4929 - val_precision: 0.1820 - val_recall: 0.6024
Epoch 3/10
1179/1179 [==============================] - 26s 22ms/step - loss: 2.0584 -
accuracy: 0.5304 - precision: 0.1876 - recall: 0.6111 - val_loss: 2.2783 -
val_accuracy: 0.4897 - val_precision: 0.2072 - val_recall: 0.5882
Epoch 4/10
1179/1179 [==============================] - 27s 23ms/step - loss: 1.9470 -
accuracy: 0.5508 - precision: 0.2081 - recall: 0.6158 - val_loss: 2.3865 -
val_accuracy: 0.4813 - val_precision: 0.2126 - val_recall: 0.5743
Epoch 5/10
1179/1179 [==============================] - 27s 23ms/step - loss: 1.8516 -
accuracy: 0.5670 - precision: 0.2355 - recall: 0.6204 - val_loss: 2.4991 -
val_accuracy: 0.4712 - val_precision: 0.2309 - val_recall: 0.5562
Epoch 6/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.7783 -
accuracy: 0.5789 - precision: 0.2602 - recall: 0.6217 - val_loss: 2.6032 -
val_accuracy: 0.4620 - val_precision: 0.2550 - val_recall: 0.5381
Epoch 7/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.7199 -
accuracy: 0.5876 - precision: 0.2885 - recall: 0.6224 - val_loss: 2.7287 -
val_accuracy: 0.4516 - val_precision: 0.2773 - val_recall: 0.5217
Epoch 8/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.6743 -
accuracy: 0.5946 - precision: 0.3116 - recall: 0.6224 - val_loss: 2.9187 -
val_accuracy: 0.4551 - val_precision: 0.2748 - val_recall: 0.5208
Epoch 9/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.6380 -
accuracy: 0.5993 - precision: 0.3264 - recall: 0.6230 - val_loss: 3.0364 -
val_accuracy: 0.4532 - val_precision: 0.2771 - val_recall: 0.5187
Epoch 10/10
1179/1179 [==============================] - 26s 22ms/step - loss: 1.6074 -
accuracy: 0.6036 - precision: 0.3469 - recall: 0.6231 - val_loss: 3.1000 -
val_accuracy: 0.4496 - val_precision: 0.2954 - val_recall: 0.5100
328/328 [==============================] - 2s 5ms/step - loss: 3.0211 -
accuracy: 0.4455 - precision: 0.2918 - recall: 0.5126
```

7 Compare and analyze models

[37]:
```
model_metrics = pd.DataFrame.from_records([
    seq_1_metrics, seq_2_metrics, lstm_1_metrics, lstm_2_metrics, lstm_3_metrics
])
model_metrics
```

[37]:
```
                             model_name  train_loss  train_acc  \
0                  sequential_model_basic    1.392291   0.646750
1  sequential_model_basic_big_embedding    1.332598   0.661757
2                        lstm_model_basic    1.230588   0.697850
3        lstm_model_basic_big_embedding    1.219402   0.700601
4                        lstm_model_large    1.607377   0.603603

   train_precision  train_recall  train_f1  test_loss  test_acc  \
0         0.356972      0.660458  0.463452   3.358078  0.415263
1         0.305888      0.675704  0.421132   3.214554  0.416026
2         0.293503      0.741819  0.420596   2.791766  0.472271
3         0.336535      0.721972  0.459079   2.982210  0.470816
4         0.346931      0.623052  0.445690   3.021123  0.445521

   test_precision  test_recall   test_f1  val_loss   val_acc  val_precision  \
0        0.264149     0.526249  0.351742  3.336539  0.433659       0.264149
1        0.258409     0.515129  0.344170  3.207735  0.430319       0.258409
2        0.244146     0.626211  0.351320  2.882399  0.477509       0.244146
3        0.265722     0.588221  0.366075  3.074434  0.474406       0.265722
4        0.291802     0.512600  0.371898  3.100049  0.449648       0.291802

   val_recall    val_f1
0    0.526249  0.356605
1    0.515129  0.347401
2    0.626211  0.349655
3    0.588221  0.361052
4    0.512600  0.374125
```

Each model has metrics for loss, accuracy, precision, recall, and F_1 score for the train, test, and validation sets. We can see that the `lstm_model_large` performed the best when compared in F_1 score to the others. Even though the accuracy is about 3% lower on the test set than the next best performing model (`lstm_model_basic`), we accept this tradeoff for better classification performance given the imblanced dataset.

If we wanted to perform better, we may try using pretrained word embeddings to capture semantic word meanings in order to boost classification performance.

Overall, the problem of predicting categories from just a headline proved to be hard to improve performance on, and a larger model does better in F_1 metrics while sometimes sacrificing accuracy.