CS4395 Portfolio Text Classification Assignment

Shane Arwood

December 4, 2022

This notebook uses a text dataset, each of which is associated with a particular emotion, like worry or sadness. With the dataset of texts and sentiments, Keras, text classification, and deep learning models are used to attempt to classify the texts into their sentiments.

```
pip install tensorflow==2.7.0
```

```
import numpy as np
import pandas as pd
np.random.seed(1234)
import tensorflow as tf
import nltk
nltk.download('stopwords')
from tensorflow import keras
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models, preprocessing
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
nltk.download('stopwords')
nltk.download('punkt')
```

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

Step 1: Dataset setup

```
# read file
texts_data = pd.read_csv('tweet_emotions.csv', header=0, usecols=['sentiment','conten
sentiment_data= texts_data[['sentiment']].astype('category')
texts_data.head()
```

	sentiment	content
0	empty	@tiffanylue i know i was listenin to bad habi...
1	sadness	Layin n bed with a headache ughhhh...waitin o...
2	sadness	Funeral ceremony...gloomy friday...
3	enthusiasm	wants to hang out with friends SOON!
4	neutral	@dannycastillo We want to trade with someone w...

```
stopwords = stopwords.words('english')

# remove stopwords
for i in range(0, 7999):
  tokens = word_tokenize(texts_data.loc[i, 'content'])
  tokens = [t.lower() for t in tokens if t.isalpha()]
  tokens = [t for t in tokens if t not in stopwords]
  texts_data.loc[i, 'text'] = ' '.join(tokens)


# Split texts_data into train and test
i = np.random.rand(len(texts_data)) < 0.8
train = texts_data[i]
test = texts_data[~i]


# Graph of distribution of target classes
print("Distribution of target classes")
print(sentiment_data.value_counts())
```

```
Distribution of target classes
sentiment
worry         2516
sadness       1771
neutral       1495
surprise       443
hate           439
happiness      361
love           293
fun            176
relief         166
empty          150
enthusiasm     102
boredom         54
anger           34
dtype: int64
```

Description of dataset and what the model should predict: The dataset contains various short texts, each with a different sentiment. The csv file's first column is the sentiment, and the second column is the The goal of the model is to be able to predict the sentiment expressed in a certain text given its contents.

Step 2: Sequential model

```
# number of classes and parameters
num_labels = 13
vocab_size = 25000
batch_size = 1000

# Fit tokenizer on content (training data)
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.content)

x_train = tokenizer.texts_to_matrix(train.content, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.content, mode='tfidf')

# Encode sentiments as they are strings
encoder = LabelEncoder()
encoder.fit(train.sentiment)
y_train = encoder.transform(train.sentiment)
y_test = encoder.transform(test.sentiment)


# create and fit model
seq_model = models.Sequential(tf.keras.layers.BatchNormalization())
seq_model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', act
seq_model.add(layers.Dense(40, kernel_initializer='normal', activation='sigmoid'))
layers.Dropout(.5) # increase dropout due to overfitting
seq_model.add(layers.Dense(13, kernel_initializer='normal', activation='softmax'))

seq_model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

seq_history = seq_model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/30
6/6 [==============================] - 6s 633ms/step - loss: 2.4931 - accuracy
Epoch 2/30
6/6 [==============================] - 2s 391ms/step - loss: 2.3089 - accuracy
Epoch 3/30
6/6 [==============================] - 2s 366ms/step - loss: 2.1525 - accuracy
Epoch 4/30
6/6 [==============================] - 2s 361ms/step - loss: 2.0608 - accuracy
Epoch 5/30
6/6 [==============================] - 2s 362ms/step - loss: 2.0044 - accuracy
Epoch 6/30
6/6 [==============================] - 2s 362ms/step - loss: 1.9693 - accuracy
```

```
Epoch 7/30
6/6 [==============================] - 2s 372ms/step - loss: 1.9435 - accuracy
Epoch 8/30
6/6 [==============================] - 2s 365ms/step - loss: 1.9230 - accuracy
Epoch 9/30
6/6 [==============================] - 2s 371ms/step - loss: 1.9001 - accuracy
Epoch 10/30
6/6 [==============================] - 2s 354ms/step - loss: 1.8736 - accuracy
Epoch 11/30
6/6 [==============================] - 2s 353ms/step - loss: 1.8445 - accuracy
Epoch 12/30
6/6 [==============================] - 2s 366ms/step - loss: 1.8113 - accuracy
Epoch 13/30
6/6 [==============================] - 2s 365ms/step - loss: 1.7740 - accuracy
Epoch 14/30
6/6 [==============================] - 2s 365ms/step - loss: 1.7318 - accuracy
Epoch 15/30
6/6 [==============================] - 2s 361ms/step - loss: 1.6851 - accuracy
Epoch 16/30
6/6 [==============================] - 2s 356ms/step - loss: 1.6316 - accuracy
Epoch 17/30
6/6 [==============================] - 2s 353ms/step - loss: 1.5753 - accuracy
Epoch 18/30
6/6 [==============================] - 2s 358ms/step - loss: 1.5148 - accuracy
Epoch 19/30
6/6 [==============================] - 2s 364ms/step - loss: 1.4475 - accuracy
Epoch 20/30
6/6 [==============================] - 2s 368ms/step - loss: 1.3735 - accuracy
Epoch 21/30
6/6 [==============================] - 2s 359ms/step - loss: 1.2975 - accuracy
Epoch 22/30
6/6 [==============================] - 2s 361ms/step - loss: 1.2213 - accuracy
Epoch 23/30
6/6 [==============================] - 2s 365ms/step - loss: 1.1475 - accuracy
Epoch 24/30
6/6 [==============================] - 2s 421ms/step - loss: 1.0788 - accuracy
Epoch 25/30
6/6 [==============================] - 3s 543ms/step - loss: 1.0121 - accuracy
Epoch 26/30
6/6 [==============================] - 3s 449ms/step - loss: 0.9523 - accuracy
Epoch 27/30
6/6 [==============================] - 2s 368ms/step - loss: 0.8951 - accuracy
Epoch 28/30
6/6 [==============================] - 2s 363ms/step - loss: 0.8446 - accuracy
Epoch 29/30
```

```
# evaluate
score = seq_model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
```

```
2/2 [==============================] - 0s 55ms/step - loss: 2.0156 - accuracy: 0
Accuracy:  0.31160768866539
```

Step 3: Different architecture, CNN

```
max_features = 10000
maxlen = 500
batch_size = 32

# Proprocess data
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)


# create CNN model
cnn_model = models.Sequential()
cnn_model.add(layers.Embedding(max_features, 13))
cnn_model.add(layers.Conv1D(32, 13, activation='relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.Conv1D(32, 13, activation='relu'))
cnn_model.add(layers.GlobalMaxPooling1D())
cnn_model.add(layers.Dense(13))


# compile model

cnn_model.compile(optimizer="adam",
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


history = cnn_model.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
40/40 [==============================] - 271s 7s/step - loss: 3.4723 - accuracy:
Epoch 2/10
40/40 [==============================] - 263s 7s/step - loss: 2.5649 - accuracy:
Epoch 3/10
40/40 [==============================] - 261s 7s/step - loss: 2.5649 - accuracy:
Epoch 4/10
40/40 [==============================] - 258s 6s/step - loss: 2.5649 - accuracy:
Epoch 5/10
40/40 [==============================] - 271s 7s/step - loss: 2.5649 - accuracy:
Epoch 6/10
40/40 [==============================] - 264s 7s/step - loss: 2.5649 - accuracy:
Epoch 7/10
40/40 [==============================] - 272s 7s/step - loss: 2.5649 - accuracy:
Epoch 8/10
40/40 [==============================] - 263s 7s/step - loss: 2.5649 - accuracy:
```

```
Epoch 9/10
40/40 [==============================] - 261s 7s/step - loss: 2.5649 - accuracy:
Epoch 10/10
40/40 [==============================] - 264s 7s/step - loss: 2.5649 - accuracy:
```

```
# evaluate
score = cnn_model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
```

```
51/51 [==============================] - 18s 348ms/step - loss: 2.5649 - accurac
Accuracy:  0.3060210943222046
```

Step 4: Embedding

```
# Divide into train/validation/test
val_split = 0.2
test_split = 0.8
val_split = int(val_split * len(texts_data))
test_split = int(test_split * len(texts_data))

val_samples = texts_data.text[:val_split]
val_labels = texts_data.sentiment[:val_split]

train_samples = texts_data.text[val_split:test_split]
train_labels = texts_data.sentiment[val_split:test_split]

test_samples = texts_data.text[test_split:]
test_labels = texts_data.sentiment[test_split:]


# Create text vectorizer
vectorizer = TextVectorization(max_tokens=25000, output_sequence_length=200)
text_ds = tf.data.Dataset.from_tensor_slices(train_samples).batch(128)
vectorizer.adapt(text_ds)


vocabulary = vectorizer.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))


# create embedding layer
EMBEDDING_DIM = 128
MAX_SEQUENCE_LENGTH = 200

embedding_layer = layers.Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            input_length=MAX_SEQUENCE_LENGTH)
```

```
# add layers
int_sequences_input = keras.Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Conv1D(128, 13, activation="relu")(embedded_sequences)
x = layers.MaxPooling1D(13)(x)
x = layers.Conv1D(128, 13, activation="relu")(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dropout(0.5)(x)
preds = layers.Dense(13, activation="softmax")(x)
embed_model = keras.Model(int_sequences_input, preds)


x_train = vectorizer(np.array([[s] for s in train_samples])).numpy()
x_val = vectorizer(np.array([[s] for s in val_samples])).numpy()

# encode y values to be ints compatible with x
y_train = encoder.transform(np.array(train_labels))
y_val = encoder.transform(np.array(val_labels))


embed_model.compile(
    loss="sparse_categorical_crossentropy", optimizer="rmsprop", metrics=["acc"])
embed_model.fit(x_train, y_train, batch_size=128, epochs=20, validation_data=(x_val,
```

```
Epoch 1/20
38/38 [==============================] - 36s 914ms/step - loss: 1.9666 - acc: 0.
Epoch 2/20
38/38 [==============================] - 37s 973ms/step - loss: 1.5457 - acc: 0.
Epoch 3/20
38/38 [==============================] - 41s 1s/step - loss: 1.2924 - acc: 0.548
Epoch 4/20
38/38 [==============================] - 33s 857ms/step - loss: 1.0923 - acc: 0.
Epoch 5/20
38/38 [==============================] - 38s 998ms/step - loss: 0.8924 - acc: 0.
Epoch 6/20
38/38 [==============================] - 36s 944ms/step - loss: 0.6898 - acc: 0.
Epoch 7/20
38/38 [==============================] - 32s 839ms/step - loss: 0.5380 - acc: 0.
Epoch 8/20
38/38 [==============================] - 32s 834ms/step - loss: 0.4372 - acc: 0.
Epoch 9/20
38/38 [==============================] - 38s 1s/step - loss: 0.3612 - acc: 0.880
Epoch 10/20
38/38 [==============================] - 31s 827ms/step - loss: 0.2963 - acc: 0.
Epoch 11/20
38/38 [==============================] - 36s 940ms/step - loss: 0.2484 - acc: 0.
Epoch 12/20
38/38 [==============================] - 40s 1s/step - loss: 0.2158 - acc: 0.929
Epoch 13/20
38/38 [==============================] - 34s 897ms/step - loss: 0.1866 - acc: 0.
Epoch 14/20
38/38 [==============================] - 31s 821ms/step - loss: 0.1690 - acc: 0.
```

```
Epoch 15/20
38/38 [==============================] - 31s 822ms/step - loss: 0.1429 - acc: 0.
Epoch 16/20
38/38 [==============================] - 31s 827ms/step - loss: 0.1347 - acc: 0.
Epoch 17/20
38/38 [==============================] - 32s 853ms/step - loss: 0.1154 - acc: 0.
Epoch 18/20
38/38 [==============================] - 37s 978ms/step - loss: 0.1030 - acc: 0.
Epoch 19/20
38/38 [==============================] - 44s 1s/step - loss: 0.0946 - acc: 0.972
Epoch 20/20
38/38 [==============================] - 31s 829ms/step - loss: 0.0826 - acc: 0.
<keras.callbacks.History at 0x7f532ee86110>
```

Analysis on performance of the models

The accuracy of the embedding approach was highest, followed by the sequential model and the other different architecture, CNN. The embedding approach reached 0.97, the sequential reached 0.84, and the CNN only got to around 0.3. However, the validation accuracies for all three approaches were low, hovering around 0.2 to 0.3. This seems to be due to overfitting. I attempted to use batch normalization, shuffling the data to ensure and even distribution in test and train, and adding dropouts between layers to mitigate this. Preprocessing the data even more could be tried as well, since these approaches only caused small improvements. Also, due to the speed constraints of google colab, using the entire dataset was not feasible. It had to be limited to around 8,000 entries, which did not seem like enough to create a well fitting model, especially becuase the distributions of the sentiments (the target classes) were so uneven in the sample I took.

There was particularly difficulty in trying to improve the CNN model simply due to the time it took for the model to fit; trying an RNN model simply took too long as well so I was unable to fully implement it at all. For the CNN model, it took several minutes for each epoch, so it was only feasible to have 10 epochs. This likely played a large role in its extremely low accuracy compared to the others. The results of the embedding approach were most promising, as they incorporated validation sets as well.