Text Classification

CS 4395.001: Human Language Technologies

Ryan Dimaranan (RTD180003) & Hannah Valena (HCV180000)

1. Dataset

We are using the following text classification dataset from Kaggle: Twitter Tweets Sentiment Dataset. This dataset has 27,500 tweets labeled according to their sentiment. Tweets are labeled as either netural, positive, or negative.

Our model should be able to predict the sentiment of a given tweet.

```
import pandas as pd

# load dataset
url = 'https://raw.githubusercontent.com/hvalena/nlp-portfolio/main/11-TextClassification/
df = pd.read_csv(url, header=0, usecols=['text', 'sentiment'])
print('rows and columns:', df.shape)

# change column types
df['sentiment'] = df.sentiment.astype('category')
df['text'] = df.text.astype('string')

# drop null values
df = df.dropna()

df.head()
```

```
rows and columns: (27481, 2)
```

	text	sentiment
0	I`d have responded, if I were going	neutral
1	Sooo SAD I will miss you here in San Diego!!!	negative
2	my boss is bullying me...	negative
3	what interview! leave me alone	negative
4	Sons of ****, why couldn`t they put them on t...	negative

```
import nltk
nltk.download('stopwords', quiet=True)
from nltk.corpus import stopwords
```

! 11s completed at 9:03 PM

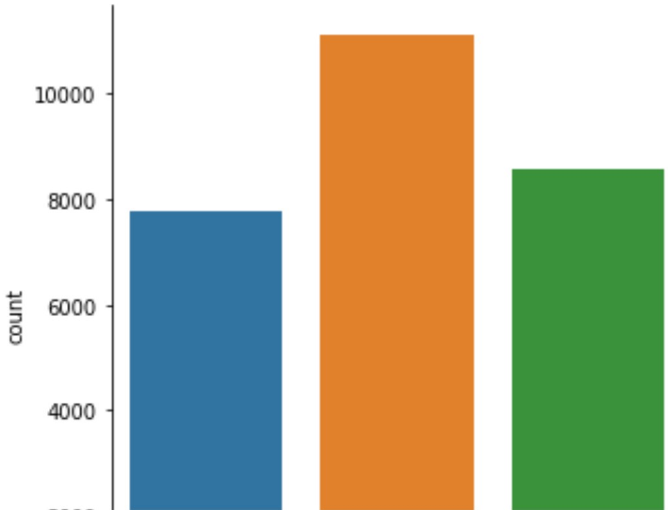
```
# perform some preprocessing
stop_words = stopwords.words('english')
df['text'] = df['text'].apply(lambda x: ' '.join([word.lower() for word in x.split() if wor
df.head()
```

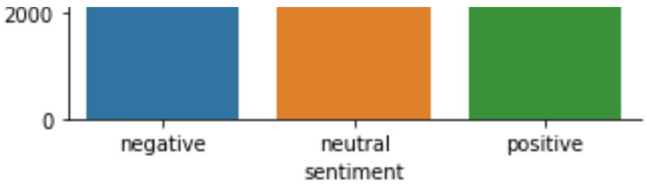
	text	sentiment
0	i`d responded, i going	neutral
1	sooo sad i miss san diego!!!	negative
2	boss bullying me...	negative
3	interview! leave alone	negative
4	sons ****, couldn`t put releases already bought	negative

```
# split df into train and test
import numpy as np

np.random.seed(1234)

i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (21997, 2)
test data size:  (5483, 2)
```

```
import seaborn as sb

# create a graph showing the distribution of the target classes
graph = sb.catplot(x="sentiment", kind="count", data=df)
```





2. Sequential Model

```
from keras.preprocessing.text import Tokenizer

vocab_size = 25000

# fit tokenizer on training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.text)


# convert to numpy matrix
x_train = tokenizer.texts_to_matrix(train.text, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.text, mode='tfidf')
```

```
from sklearn.preprocessing import LabelEncoder

# encode, normalize target values
encoder = LabelEncoder()
encoder.fit(train.sentiment)
y_train = encoder.transform(train.sentiment)
y_test = encoder.transform(test.sentiment)
```

```
# check shape
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("train first 10 labels:", y_train[:10])
print("test first 10 labels:", y_test[:10])
```

```
     train shapes: (21997, 25000) (21997,)
     test shapes: (5483, 25000) (5483,)
     train first 10 labels: [1 0 0 0 0 1 2 1 2 0]
     test first 10 labels: [1 1 2 1 1 2 0 1 2 1]
```

```
from tensorflow.keras import layers, models

# create a sequential model
model = models.Sequential()
model.add(layers.Dense(16, input_dim=vocab_size, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
# compile
model.compile(loss='mse',
              optimizer='rmsprop',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                400016

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 400,033
Trainable params: 400,033
Non-trainable params: 0
_________________________________________________________________
```

```
# train model
model.fit(x_train, y_train,
            batch_size=128,
            epochs=10,
            verbose=1,
            validation_split=0.1)
```

```
Epoch 1/10
198/198 [==============================] - 8s 36ms/step - loss: 0.5437 - accuracy: 0.
Epoch 2/10
198/198 [==============================] - 8s 41ms/step - loss: 0.4379 - accuracy: 0.
Epoch 3/10
198/198 [==============================] - 6s 31ms/step - loss: 0.4079 - accuracy: 0.
Epoch 4/10
198/198 [==============================] - 6s 29ms/step - loss: 0.3908 - accuracy: 0.
Epoch 5/10
198/198 [==============================] - 6s 32ms/step - loss: 0.3787 - accuracy: 0.
Epoch 6/10
198/198 [==============================] - 6s 31ms/step - loss: 0.3700 - accuracy: 0.
Epoch 7/10
198/198 [==============================] - 6s 31ms/step - loss: 0.3632 - accuracy: 0.
Epoch 8/10
198/198 [==============================] - 7s 33ms/step - loss: 0.3578 - accuracy: 0.
Epoch 9/10
198/198 [==============================] - 6s 33ms/step - loss: 0.3531 - accuracy: 0.
Epoch 10/10
198/198 [==============================] - 6s 32ms/step - loss: 0.3492 - accuracy: 0.
```

```
# evaluate
score = model.evaluate(x_test, y_test, batch_size=100, verbose=1)
print('Accuracy: ', score[1])
print('Accuracy: ', score[1])
```

```
55/55 [==============================] - 1s 10ms/step - loss: 0.5057 - accuracy: 0.56
Accuracy:  0.5013678669929504
```

3. RNN & CNN

```
from tensorflow.keras import layers, models

# RNN
model = models.Sequential()
model.add(layers.Embedding(vocab_size, 64))
model.add(layers.SimpleRNN(64))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train,
        y_train,
        epochs=10,
        batch_size=128,
        validation_split=0.2)

score = model.evaluate(x_test, y_test, batch_size=100, verbose=1)
print('Accuracy: ', score[1])
```

```
from tensorflow.keras import layers, models

# CNN
model = models.Sequential()
model.add(layers.Embedding(25000, 128,))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))


model.summary()

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train,
          y_train,
          epochs=10,
          batch_size=128,
          validation_split=0.2)

score = model.evaluate(x_test, y_test, batch_size=100, verbose=1)
print('Accuracy: ', score[1])
```

4. Embeddings

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# split test and validation
X_test, X_valid, y_test, y_valid = train_test_split(X, y, test_size=0.2,
                                                    train_size=0.8,
                                                    random_state=1234)
# train on all data
X_train = X
y_train = y

# labels to integers
encoder = LabelEncoder()
encoder.fit(y)

y_valid = encoder.transform(y_valid)
y_test = encoder.transform(y_test)
y_train = encoder.transform(y_train)

# vectorize init
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
import tensorflow as tf

vectorizer = TextVectorization(max_tokens=20000, output_sequence_length=200)
text_ds = tf.data.Dataset.from_tensor_slices(X_train).batch(128)
vectorizer.adapt(text_ds)

# get vocab
voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))


from tensorflow.keras import layers
from tensorflow import keras

EMBEDDING_DIM = 128
MAX_SEQUENCE_LENGTH = 200
```

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

```
import numpy as np

# vectorize data
x_train = vectorizer(np.array([[s] for s in X_train])).numpy()
x_val = vectorizer(np.array([[s] for s in X_valid])).numpy()
x_test  = vectorizer(np.array([[s] for s in X_test])).numpy()

y_train = np.array(y_train)
y_val = np.array(y_valid)
y_test = np.array(y_test)
```

```
# init model and add layers
int_sequences_input = keras.Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Conv1D(128, 5, activation="relu")(embedded_sequences)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dropout(0.5)(x)

preds = layers.Dense(len(y), activation="softmax")(x)
model = keras.Model(int_sequences_input, preds)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 embedding (Embedding)       (None, None, 128)         2560128

 conv1d (Conv1D)             (None, None, 128)         82048

 max_pooling1d (MaxPooling1D  (None, None, 128)        0
 )

 conv1d_1 (Conv1D)           (None, None, 128)         82048

 max_pooling1d_1 (MaxPooling  (None, None, 128)        0
 1D)

 conv1d_2 (Conv1D)           (None, None, 128)         82048
 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dense (Dense)              (None, 128)              16512

 dropout (Dropout)          (None, 128)              0

 dense_1 (Dense)            (None, 27480)            3544920

=================================================================
Total params: 6,367,704
Trainable params: 6,367,704
Non-trainable params: 0
_________________________________________________________________
```

```
# compile model and train
model.compile(
    loss="sparse_categorical_crossentropy", optimizer="rmsprop", metrics=["acc"]
)

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2)

model.fit(x_train, y_train, batch_size=128, epochs=15, validation_data=(x_val, y_val), cal
```

```
Epoch 1/15
215/215 [==============================] - 115s 530ms/step - loss: 1.2898 - acc: 0.37
Epoch 2/15
215/215 [==============================] - 112s 521ms/step - loss: 1.0977 - acc: 0.39
Epoch 3/15
215/215 [==============================] - 105s 489ms/step - loss: 1.0040 - acc: 0.49
Epoch 4/15
215/215 [==============================] - 100s 464ms/step - loss: 0.8088 - acc: 0.66
Epoch 5/15
215/215 [==============================] - 104s 485ms/step - loss: 0.6864 - acc: 0.74
Epoch 6/15
215/215 [==============================] - 99s 461ms/step - loss: 0.5591 - acc: 0.788
Epoch 7/15
215/215 [==============================] - 101s 469ms/step - loss: 0.4311 - acc: 0.84
Epoch 8/15
215/215 [==============================] - 99s 461ms/step - loss: 0.3336 - acc: 0.885
Epoch 9/15
215/215 [==============================] - 100s 463ms/step - loss: 0.2482 - acc: 0.91
Epoch 10/15
215/215 [==============================] - 99s 459ms/step - loss: 0.1807 - acc: 0.940
Epoch 11/15
215/215 [==============================] - 98s 455ms/step - loss: 0.1304 - acc: 0.959
Epoch 12/15
215/215 [==============================] - 95s 444ms/step - loss: 0.0966 - acc: 0.969
Epoch 13/15
215/215 [==============================] - 97s 450ms/step - loss: 0.0721 - acc: 0.978
Epoch 14/15
215/215 [==============================] - 96s 445ms/step - loss: 0.0583 - acc: 0.982
Epoch 15/15
```

```
     215/215 [==============================] - 96s 448ms/step - loss: 0.0451 - acc: 0.986
     <keras.callbacks.History at 0x7f4769b86190>
```

```
# test and evaluate model
score = model.evaluate(x_test, y_test, batch_size=128, verbose=1)
print('Accuracy: ', score[1])
```

```
     172/172 [==============================] - 30s 172ms/step - loss: 0.3378 - acc: 0.879
     Accuracy:  0.8789574503898621
```

```
!wget --no-check-certificate http://nlp.stanford.edu/data/glove.6B.zip
```

```
!unzip glove.6B.zip
```

```
import os
import numpy as np

# get glove embeddings
path_to_glove_file = os.path.join(
  "glove.6B.200d.txt"
)

num_tokens = len(voc) + 2
embedding_dim = 200
hits = 0
misses = 0

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
```

```
from tensorflow.keras.layers import Embedding

# initialize embedding layer with pretrained embeddings
embedding_layer = Embedding(
    num_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
)

# compile and train with early stopping
model.compile(
    loss="sparse_categorical_crossentropy", optimizer="rmsprop", metrics=["acc"]
)

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2)

model.fit(x_train, y_train, batch_size=128, epochs=15, validation_data=(x_val, y_val), cal
```

```
Epoch 1/15
215/215 [==============================] - 87s 399ms/step - loss: 0.4466 - acc: 0.826
Epoch 2/15
215/215 [==============================] - 88s 409ms/step - loss: 0.3826 - acc: 0.851
Epoch 3/15
215/215 [==============================] - 85s 398ms/step - loss: 0.3388 - acc: 0.873
Epoch 4/15
215/215 [==============================] - 88s 408ms/step - loss: 0.3086 - acc: 0.885
Epoch 5/15
215/215 [==============================] - 85s 398ms/step - loss: 0.2750 - acc: 0.899
Epoch 6/15
215/215 [==============================] - 88s 408ms/step - loss: 0.2555 - acc: 0.906
Epoch 7/15
215/215 [==============================] - 93s 435ms/step - loss: 0.2287 - acc: 0.918
Epoch 8/15
215/215 [==============================] - 86s 401ms/step - loss: 0.2024 - acc: 0.928
Epoch 9/15
215/215 [==============================] - 88s 407ms/step - loss: 0.1913 - acc: 0.932
Epoch 10/15
215/215 [==============================] - 93s 432ms/step - loss: 0.1808 - acc: 0.937
Epoch 11/15
215/215 [==============================] - 87s 403ms/step - loss: 0.1685 - acc: 0.941
Epoch 12/15
215/215 [==============================] - 85s 398ms/step - loss: 0.1513 - acc: 0.947
Epoch 13/15
215/215 [==============================] - 87s 402ms/step - loss: 0.1521 - acc: 0.948
Epoch 14/15
215/215 [==============================] - 85s 397ms/step - loss: 0.1369 - acc: 0.953
Epoch 15/15
215/215 [==============================] - 87s 405ms/step - loss: 0.1313 - acc: 0.953
<keras.callbacks.History at 0x7fdfc0cb9940>
```

```
# evaluate model
score = model.evaluate(x_test, y_test, batch_size=128, verbose=1)
print('Accuracy: ', score[1])
```

```
172/172 [==============================] - 29s 167ms/step - loss: 0.2370 - acc: 0.913
Accuracy:  0.9137099981307983
```