

# Assignment 9: Text Classification with Deep Learning

Brendan Lim

CS4395.001

In [88]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import tensorflow as tf
from tensorflow.keras import layers, models, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import matplotlib.pyplot as plt

import os
from pathlib import Path

#code that came with kaggle
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

/kaggle/input/news-headlines-dataset-for-sarcasm-detection/Sarcasm_Headlines_Dataset_v2.
json
/kaggle/input/news-headlines-dataset-for-sarcasm-detection/Sarcasm_Headlines_Dataset.js
on
```

## Sarcasm Headline Dataset

This sarcasm dataset takes headlines from the Onion (sarcastic) and the Huffington Post (serious) to determine whether a headline is sarcastic or not. The model must be able to predict between sarcastic and serious headlines, although I figure that it will veer more towards being able to predict the writing styles of the two news companies.

Citation:

1. Misra, Rishabh and Prahal Arora. "Sarcasm Detection using News Headlines Dataset." AI Open (2023).
2. Misra, Rishabh and Jigyasa Grover. "Sculpting Data for ML: The first act of Machine Learning." ISBN 9798585463570 (2021).

In [89]:

```
#read in json file using pandas
dataset_path = Path("/kaggle/input/news-headlines-dataset-for-sarcasm-detection/Sarcasm

json_df = pd.read_json(dataset_path, lines= True)
```

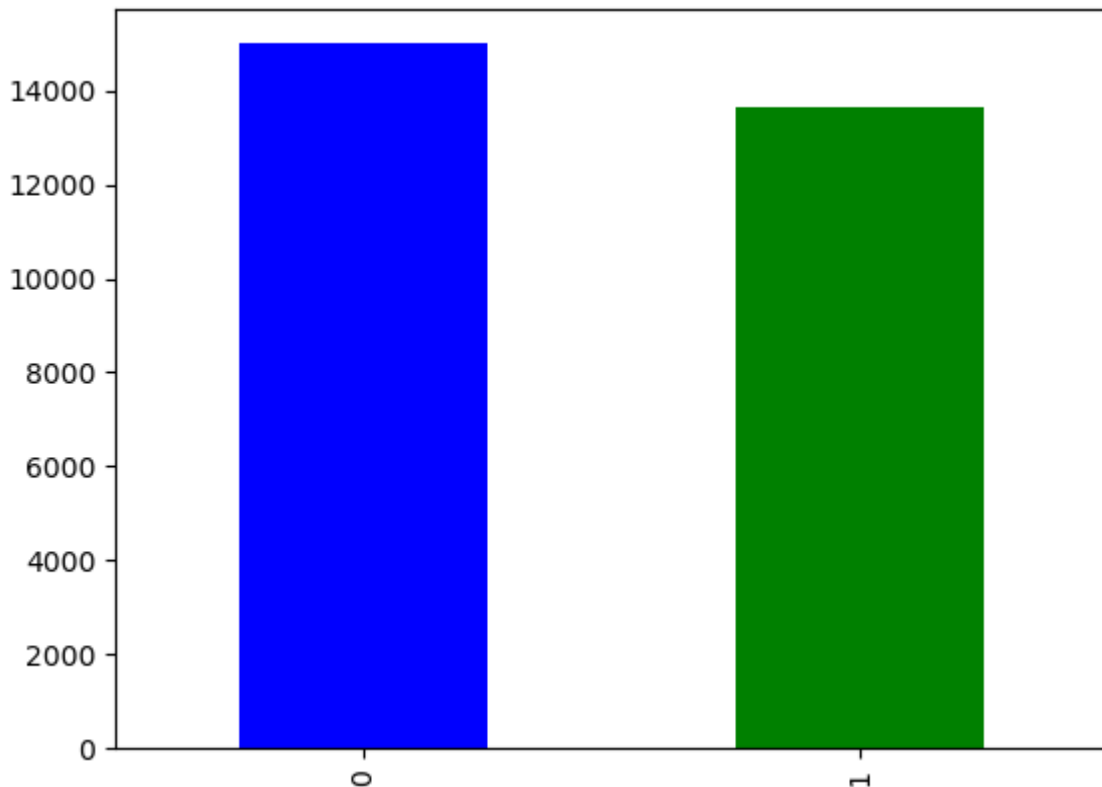
```
dataset = json_df[["headline", "is_sarcastic"]]
print(dataset)
```

	headline	is_sarcastic
0	thirtysomething scientists unveil doomsday clo...	1
1	dem rep. totally nails why congress is falling...	0
2	eat your veggies: 9 deliciously different recipes	0
3	inclement weather prevents liar from getting t...	1
4	mother comes pretty close to using word 'strea...	1
...	...	...
28614	jews to celebrate rosh hashasha or something	1
28615	internal affairs investigator disappointed con...	1
28616	the most beautiful acceptance speech this week...	0
28617	mars probe destroyed by orbiting spielberg-gat...	1
28618	dad clarifies this not a food stop	1

[28619 rows x 2 columns]

```
In [90]: dataset['is_sarcastic'].value_counts().plot(kind='bar', color = ["b", 'g'])
```

Out[90]: <AxesSubplot:>



```
In [91]: #divide into train/valid/test
train_split = 0.8
test_split = 0.2
#validation will be 25% of training dataset, or 20% of whole dataset
valid_split = 0.25

x_train, x_test, y_train, y_test = train_test_split(dataset.headline, dataset.is_sarcas
x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size= vali
print(x_train.shape)
```

```
print(x_valid.shape)
print(x_test.shape)
```

```
(17171,)
(5724,)
(5724,)
```

In [92]:

```
#articles headlines are expected to have higher diversity of speech
text_vect_layer = tf.keras.layers.TextVectorization(
    max_tokens = 10000,
    output_mode = 'int',
    output_sequence_length=50
)
text_vect_layer.adapt(x_train)
```

## Simple Sequential Model

The simple sequential did not perform very well, even after being given many different variations of dense neural layers. I tried using different activation functions, but that did not seem to have a significant effect. This model seems to be the stepping stone between ML neural networks and the deep learnign models we've discussed earlier.

In [120...

```
seq_model = models.Sequential()
seq_model.add(tf.keras.Input(shape=(1,), dtype=tf.string))
seq_model.add(text_vect_layer)
seq_model.add(layers.Dense(30, activation='relu'))
seq_model.add(layers.Dense(30, activation='relu'))
seq_model.add(layers.Dense(1, activation='sigmoid'))

seq_model.summary()

seq_model.compile(
    optimizer = 'adam',
    loss = "binary_crossentropy",
    metrics=['accuracy']
)
```

Model: "sequential\_29"

Layer (type)	Output Shape	Param #
text_vectorization_3 (TextVectorization)	(None, 50)	0
dense_42 (Dense)	(None, 30)	1530
dense_43 (Dense)	(None, 30)	930
dense_44 (Dense)	(None, 1)	31
Total params: 2,491		
Trainable params: 2,491		
Non-trainable params: 0		

In [121...

```

history = seq_model.fit(
    x_train,
    y_train,
    epochs = 10,
    batch_size = 256,
    validation_data=((x_valid, y_valid))
)

```

```

Epoch 1/10
68/68 [=====] - 1s 8ms/step - loss: 110.4009 - accuracy: 0.5004
- val_loss: 35.0019 - val_accuracy: 0.5262
Epoch 2/10
68/68 [=====] - 0s 5ms/step - loss: 30.9152 - accuracy: 0.5239
- val_loss: 23.0225 - val_accuracy: 0.5143
Epoch 3/10
68/68 [=====] - 0s 6ms/step - loss: 21.9473 - accuracy: 0.5215
- val_loss: 18.2069 - val_accuracy: 0.5101
Epoch 4/10
68/68 [=====] - 0s 5ms/step - loss: 17.5716 - accuracy: 0.5247
- val_loss: 15.1931 - val_accuracy: 0.5147
Epoch 5/10
68/68 [=====] - 0s 4ms/step - loss: 14.6262 - accuracy: 0.5275
- val_loss: 13.1573 - val_accuracy: 0.5203
Epoch 6/10
68/68 [=====] - 0s 4ms/step - loss: 12.1413 - accuracy: 0.5304
- val_loss: 11.1731 - val_accuracy: 0.5114
Epoch 7/10
68/68 [=====] - 0s 5ms/step - loss: 10.3357 - accuracy: 0.5292
- val_loss: 9.3746 - val_accuracy: 0.5236
Epoch 8/10
68/68 [=====] - 0s 5ms/step - loss: 9.0609 - accuracy: 0.5339
- val_loss: 8.4005 - val_accuracy: 0.5245
Epoch 9/10
68/68 [=====] - 0s 6ms/step - loss: 7.9575 - accuracy: 0.5346
- val_loss: 7.3633 - val_accuracy: 0.5327
Epoch 10/10
68/68 [=====] - 0s 4ms/step - loss: 7.0928 - accuracy: 0.5366
- val_loss: 6.9588 - val_accuracy: 0.5280

```

In [115...

```
#predict on test data
pred = seq_model.predict(x_test)
```

179/179 [=====] - 0s 2ms/step

In [116...

```
bool_pred = np.where(pred < 0.5, 0, 1)
print(pred[:10])
print(bool_pred[:10])

print(classification_report(y_test, bool_pred))
```

```
[[1.0000000e+00]
 [6.9303697e-01]
 [9.9301457e-01]
 [1.0000000e+00]
 [1.0534343e-05]
 [9.8170125e-01]
 [1.0000000e+00]
 [9.4150776e-01]
 [1.0000000e+00]
 [6.7488298e-05]]
```

```
[[1]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]]
```

	precision	recall	f1-score	support
0	0.55	0.51	0.53	3035
1	0.49	0.52	0.51	2689
accuracy			0.52	5724
macro avg	0.52	0.52	0.52	5724
weighted avg	0.52	0.52	0.52	5724

## Convolutional Neural Network

I've only ever used CNNs for things like images, so I was curious to see how a 1D CNN works.

Something interesting is that the test accuracy was near 100%, while the validation and test accuracy were very nearly the same at ~80%, showing an example of overfitting. I am surprised that nearly any CNN model I threw the dataset at was able to get a good starting accuracy, especially comparing it to other neural networks, which require a decent amount of fine tuning before getting good results.

I think I would have to read up more on how to choose filter/kernel size so that I can design better models rather than just doing trial-and-error.

In [97]:

```

cnn_model = models.Sequential()
cnn_model.add(tf.keras.Input(shape=(1,), dtype=tf.string))
cnn_model.add(text_vect_layer)
#10000 vocab size and 50 input length
cnn_model.add(layers.Embedding(10000,64,input_length = 50))
cnn_model.add(layers.Conv1D(64,7, activation = 'relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.Conv1D(32,3, activation = 'relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.GlobalMaxPooling1D())
cnn_model.add(layers.Dense(1))
cnn_model.summary()

```

Model: "sequential\_21"

Layer (type)	Output Shape	Param #
=====		
text_vectorization_3 (TextVectorization)	(None, 50)	0
embedding_17 (Embedding)	(None, 50, 64)	640000
conv1d_34 (Conv1D)	(None, 44, 64)	28736
max_pooling1d_31 (MaxPooling1D)	(None, 8, 64)	0
conv1d_35 (Conv1D)	(None, 6, 32)	6176
max_pooling1d_32 (MaxPooling1D)	(None, 1, 32)	0
global_max_pooling1d_10 (GlobalMaxPooling1D)	(None, 32)	0
dense_26 (Dense)	(None, 1)	33
=====		
Total params: 674,945		
Trainable params: 674,945		
Non-trainable params: 0		

In [98]:

```

#model definition
cnn_model.compile(
    optimizer = "adam",
    loss="binary_crossentropy",
    metrics= ['accuracy']
)

```

In [99]:

```

#training the model
history = cnn_model.fit(
    x_train,
    y_train,
    epochs = 20,
    batch_size = 256,
)

```

```
validation_data=((x_valid, y_valid))  
)
```

```
Epoch 1/20  
68/68 [=====] - 5s 50ms/step - loss: 0.7903 - accuracy: 0.5556  
- val_loss: 0.6822 - val_accuracy: 0.5704  
Epoch 2/20  
68/68 [=====] - 3s 41ms/step - loss: 0.6502 - accuracy: 0.7212  
- val_loss: 0.6377 - val_accuracy: 0.7121  
Epoch 3/20  
68/68 [=====] - 3s 42ms/step - loss: 0.5124 - accuracy: 0.8401  
- val_loss: 0.4888 - val_accuracy: 0.7874  
Epoch 4/20  
68/68 [=====] - 3s 43ms/step - loss: 0.3006 - accuracy: 0.9033  
- val_loss: 0.5187 - val_accuracy: 0.8176  
Epoch 5/20  
68/68 [=====] - 3s 41ms/step - loss: 0.1725 - accuracy: 0.9518  
- val_loss: 0.5863 - val_accuracy: 0.8253  
Epoch 6/20  
68/68 [=====] - 3s 41ms/step - loss: 0.1104 - accuracy: 0.9777  
- val_loss: 0.8005 - val_accuracy: 0.8249  
Epoch 7/20  
68/68 [=====] - 3s 42ms/step - loss: 0.0854 - accuracy: 0.9848  
- val_loss: 0.8318 - val_accuracy: 0.8277  
Epoch 8/20  
68/68 [=====] - 3s 43ms/step - loss: 0.0608 - accuracy: 0.9920  
- val_loss: 0.8836 - val_accuracy: 0.8305  
Epoch 9/20  
68/68 [=====] - 3s 41ms/step - loss: 0.0500 - accuracy: 0.9939  
- val_loss: 0.9395 - val_accuracy: 0.8274  
Epoch 10/20  
68/68 [=====] - 3s 42ms/step - loss: 0.0458 - accuracy: 0.9951  
- val_loss: 0.9523 - val_accuracy: 0.8248  
Epoch 11/20  
68/68 [=====] - 3s 41ms/step - loss: 0.0416 - accuracy: 0.9959  
- val_loss: 0.9817 - val_accuracy: 0.8248  
Epoch 12/20  
68/68 [=====] - 3s 47ms/step - loss: 0.0398 - accuracy: 0.9964  
- val_loss: 0.9901 - val_accuracy: 0.8248  
Epoch 13/20  
68/68 [=====] - 3s 42ms/step - loss: 0.0382 - accuracy: 0.9967  
- val_loss: 1.0106 - val_accuracy: 0.8248  
Epoch 14/20  
68/68 [=====] - 3s 41ms/step - loss: 0.0375 - accuracy: 0.9970  
- val_loss: 1.0629 - val_accuracy: 0.8258  
Epoch 15/20  
68/68 [=====] - 3s 41ms/step - loss: 0.0369 - accuracy: 0.9971  
- val_loss: 1.0742 - val_accuracy: 0.8258  
Epoch 16/20  
68/68 [=====] - 3s 43ms/step - loss: 0.0365 - accuracy: 0.9973  
- val_loss: 1.0815 - val_accuracy: 0.8272  
Epoch 17/20  
68/68 [=====] - 3s 41ms/step - loss: 0.0363 - accuracy: 0.9973  
- val_loss: 1.1005 - val_accuracy: 0.8274  
Epoch 18/20  
68/68 [=====] - 3s 41ms/step - loss: 0.0361 - accuracy: 0.9973  
- val_loss: 1.1039 - val_accuracy: 0.8283  
Epoch 19/20  
68/68 [=====] - 3s 42ms/step - loss: 0.0359 - accuracy: 0.9973  
- val_loss: 1.1090 - val_accuracy: 0.8272
```

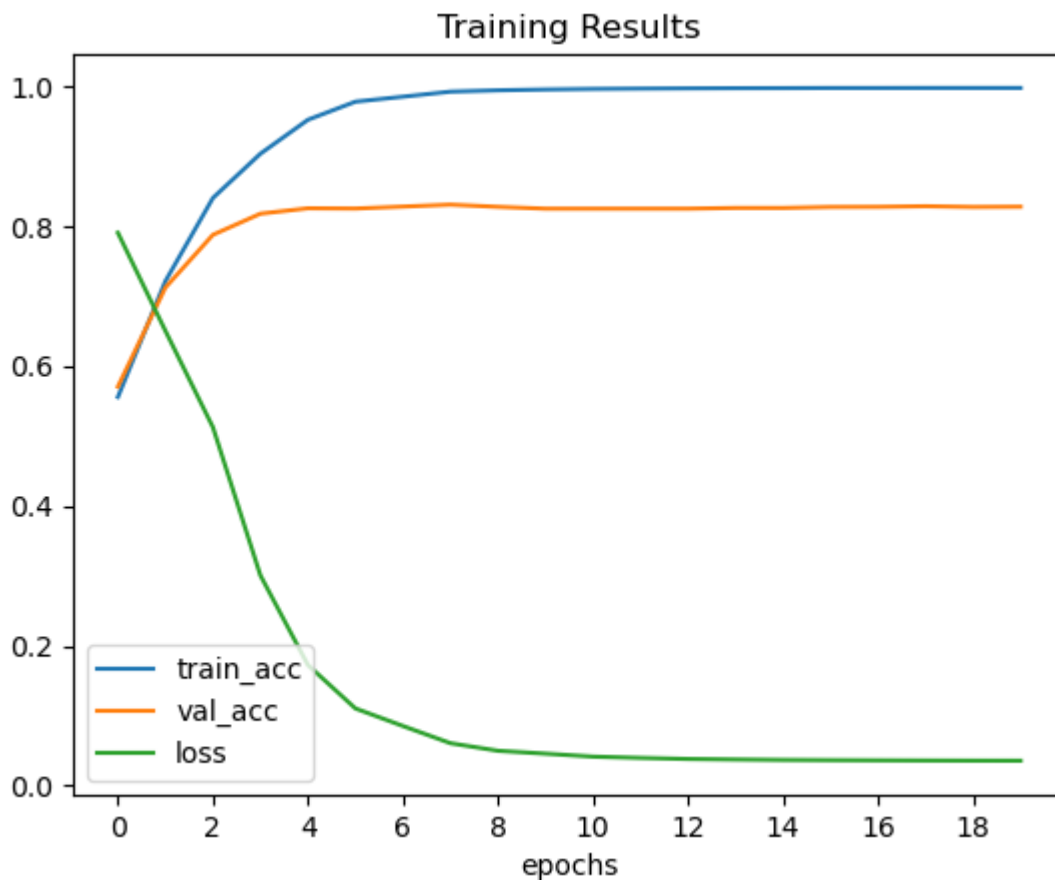
Epoch 20/20

68/68 [=====] - 3s 42ms/step - loss: 0.0358 - accuracy: 0.9973  
 - val\_loss: 1.1325 - val\_accuracy: 0.8276

In [100...

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.title("Training Results")
plt.xlabel('epochs')
plt.xticks(np.arange(0, 20, 2))
plt.legend(['train_acc', 'val_acc', 'loss'], loc = 'lower left')

plt.show()
```



In [101...

```
pred = cnn_model.predict(x_test)
```

179/179 [=====] - 1s 3ms/step

In [102...

```
print(pred.shape)
bool_pred = np.where(pred < 0.5, 0, 1)
new_bool_pred = bool_pred.reshape((y_test.shape[0],1))
new_y_test = y_test.values.reshape((y_test.shape[0],1))
print(new_bool_pred.shape)
print(new_y_test.shape)
print(classification_report(new_y_test, new_bool_pred))
```

(5724, 1)

(5724, 1)

(5724, 1)



	precision	recall	f1-score	support
0	0.84	0.84	0.84	3035
1	0.82	0.82	0.82	2689
accuracy			0.83	5724
macro avg	0.83	0.83	0.83	5724
weighted avg	0.83	0.83	0.83	5724

## Trying out different embedding approaches

Trying out a 128, 32, and 16 dimensional vector for embedding. They do not seem to produce noticeably better results. The model seems to overfit either way and have 80% accuracy for the validation and test sets.

In [103...

```
#128
cnn_model = models.Sequential()
cnn_model.add(tf.keras.Input(shape=(1,), dtype=tf.string))
cnn_model.add(text_vect_layer)
#10000 vocab size and 50 input length
cnn_model.add(layers.Embedding(10000,128,input_length = 50))
cnn_model.add(layers.Conv1D(64,7, activation = 'relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.Conv1D(32,3, activation = 'relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.GlobalMaxPooling1D())
cnn_model.add(layers.Dense(1))

cnn_model.summary()

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

history = cnn_model.fit(
    x_train,
    y_train,
    epochs = 10,
    batch_size = 256,
    validation_data=((x_valid, y_valid))
)

pred = cnn_model.predict(x_test)
bool_pred = np.where(pred < 0.5, 0, 1)
new_bool_pred = bool_pred.reshape((y_test.shape[0],1))
new_y_test = y_test.values.reshape((y_test.shape[0],1))
print(classification_report(new_y_test, new_bool_pred))
```

Model: "sequential\_22"

Layer (type)	Output Shape	Param #
text_vectorization_3 (TextV	(None, 50)	0

ectorization)

embedding_18 (Embedding)	(None, 50, 128)	1280000
conv1d_36 (Conv1D)	(None, 44, 64)	57408
max_pooling1d_33 (MaxPoolin g1D)	(None, 8, 64)	0
conv1d_37 (Conv1D)	(None, 6, 32)	6176
max_pooling1d_34 (MaxPoolin g1D)	(None, 1, 32)	0
global_max_pooling1d_11 (Gl obalMaxPooling1D)	(None, 32)	0
dense_27 (Dense)	(None, 1)	33

=====

Total params: 1,343,617

Trainable params: 1,343,617

Non-trainable params: 0

---

Epoch 1/10

68/68 [=====] - 6s 77ms/step - loss: 4.3288 - accuracy: 0.5214  
- val\_loss: 0.6864 - val\_accuracy: 0.5196

Epoch 2/10

68/68 [=====] - 5s 70ms/step - loss: 0.5836 - accuracy: 0.7538  
- val\_loss: 0.4800 - val\_accuracy: 0.7802

Epoch 3/10

68/68 [=====] - 5s 70ms/step - loss: 0.3244 - accuracy: 0.8789  
- val\_loss: 0.5268 - val\_accuracy: 0.8085

Epoch 4/10

68/68 [=====] - 5s 71ms/step - loss: 0.2072 - accuracy: 0.9493  
- val\_loss: 0.7160 - val\_accuracy: 0.8297

Epoch 5/10

68/68 [=====] - 5s 70ms/step - loss: 0.1536 - accuracy: 0.9715  
- val\_loss: 0.8629 - val\_accuracy: 0.8265

Epoch 6/10

68/68 [=====] - 5s 71ms/step - loss: 0.0923 - accuracy: 0.9864  
- val\_loss: 0.9990 - val\_accuracy: 0.8256

Epoch 7/10

68/68 [=====] - 5s 70ms/step - loss: 0.0761 - accuracy: 0.9904  
- val\_loss: 1.0438 - val\_accuracy: 0.8202

Epoch 8/10

68/68 [=====] - 5s 77ms/step - loss: 0.0608 - accuracy: 0.9931  
- val\_loss: 1.0081 - val\_accuracy: 0.8211

Epoch 9/10

68/68 [=====] - 5s 70ms/step - loss: 0.0498 - accuracy: 0.9951  
- val\_loss: 1.0436 - val\_accuracy: 0.8215

Epoch 10/10

68/68 [=====] - 5s 70ms/step - loss: 0.0461 - accuracy: 0.9954  
- val\_loss: 1.1950 - val\_accuracy: 0.8239

179/179 [=====] - 1s 4ms/step

	precision	recall	f1-score	support
0	0.84	0.85	0.84	3035
1	0.82	0.82	0.82	2689

accuracy			0.83	5724
macro avg	0.83	0.83	0.83	5724
weighted avg	0.83	0.83	0.83	5724

In [104...

```
#32
cnn_model = models.Sequential()
cnn_model.add(tf.keras.Input(shape=(1,), dtype=tf.string))
cnn_model.add(text_vect_layer)
#10000 vocab size and 50 input length
cnn_model.add(layers.Embedding(10000,32,input_length = 50))
cnn_model.add(layers.Conv1D(32,2, activation = 'relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.Conv1D(16,2, activation = 'relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.GlobalMaxPooling1D())
cnn_model.add(layers.Dense(1))

cnn_model.summary()

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

history = cnn_model.fit(
    x_train,
    y_train,
    epochs = 10,
    batch_size = 256,
    validation_data=((x_valid, y_valid))
)

pred = cnn_model.predict(x_test)
bool_pred = np.where(pred < 0.5, 0, 1)
new_bool_pred = bool_pred.reshape((y_test.shape[0],1))
print(classification_report(new_y_test, new_bool_pred))
```

Model: "sequential\_23"

Layer (type)	Output Shape	Param #
text_vectorization_3 (TextVectorization)	(None, 50)	0
embedding_19 (Embedding)	(None, 50, 32)	320000
conv1d_38 (Conv1D)	(None, 49, 32)	2080
max_pooling1d_35 (MaxPooling1D)	(None, 9, 32)	0
conv1d_39 (Conv1D)	(None, 8, 16)	1040
max_pooling1d_36 (MaxPooling1D)	(None, 1, 16)	0
global_max_pooling1d_12 (GlobalMaxPooling1D)	(None, 16)	0

obalMaxPooling1D)

dense\_28 (Dense) (None, 1) 17

```
=====
Total params: 323,137
Trainable params: 323,137
Non-trainable params: 0
```

```
Epoch 1/10
68/68 [=====] - 2s 20ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 2/10
68/68 [=====] - 1s 16ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 3/10
68/68 [=====] - 1s 16ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 4/10
68/68 [=====] - 1s 16ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 5/10
68/68 [=====] - 1s 16ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 6/10
68/68 [=====] - 1s 16ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 7/10
68/68 [=====] - 1s 16ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 8/10
68/68 [=====] - 1s 16ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 9/10
68/68 [=====] - 1s 16ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
Epoch 10/10
68/68 [=====] - 1s 17ms/step - loss: 7.3617 - accuracy: 0.5227
- val_loss: 7.4107 - val_accuracy: 0.5196
179/179 [=====] - 1s 2ms/step
```

	precision	recall	f1-score	support
0	0.53	1.00	0.69	3035
1	0.00	0.00	0.00	2689
accuracy			0.53	5724
macro avg	0.27	0.50	0.35	5724
weighted avg	0.28	0.53	0.37	5724

```
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

```
_warn_prf(average, modifier, msg_start, len(result))
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

```
_warn_prf(average, modifier, msg_start, len(result))
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

h no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))

In [105...

```
#16
cnn_model = models.Sequential()
cnn_model.add(tf.keras.Input(shape=(1,), dtype=tf.string))
cnn_model.add(text_vect_layer)
#10000 vocab size and 50 input length
cnn_model.add(layers.Embedding(10000,16,input_length = 50))
cnn_model.add(layers.Conv1D(64,7, activation = 'relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.Conv1D(32,3, activation = 'relu'))
cnn_model.add(layers.MaxPooling1D(5))
cnn_model.add(layers.GlobalMaxPooling1D())
cnn_model.add(layers.Dense(1))

cnn_model.summary()

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

history = cnn_model.fit(
    x_train,
    y_train,
    epochs = 10,
    batch_size = 256,
    validation_data=((x_valid, y_valid))
)

pred = cnn_model.predict(x_test)
bool_pred = np.where(pred < 0.5, 0, 1)
new_bool_pred = bool_pred.reshape((y_test.shape[0],1))
print(classification_report(new_y_test, new_bool_pred))
```

Model: "sequential\_24"

Layer (type)	Output Shape	Param #
=====		
text_vectorization_3 (TextVectorization)	(None, 50)	0
embedding_20 (Embedding)	(None, 50, 16)	160000
conv1d_40 (Conv1D)	(None, 44, 64)	7232
max_pooling1d_37 (MaxPooling1D)	(None, 8, 64)	0
conv1d_41 (Conv1D)	(None, 6, 32)	6176
max_pooling1d_38 (MaxPooling1D)	(None, 1, 32)	0
global_max_pooling1d_13 (GlobalMaxPooling1D)	(None, 32)	0

dense\_29 (Dense)

(None, 1)

33

=====

Total params: 173,441

Trainable params: 173,441

Non-trainable params: 0

Epoch 1/10

68/68 [=====] - 3s 25ms/step - loss: 0.8474 - accuracy: 0.5506  
- val\_loss: 0.6712 - val\_accuracy: 0.6244

Epoch 2/10

68/68 [=====] - 2s 25ms/step - loss: 0.6209 - accuracy: 0.7064  
- val\_loss: 0.5796 - val\_accuracy: 0.6964

Epoch 3/10

68/68 [=====] - 2s 22ms/step - loss: 0.4226 - accuracy: 0.8264  
- val\_loss: 0.4980 - val\_accuracy: 0.7956

Epoch 4/10

68/68 [=====] - 2s 24ms/step - loss: 0.2765 - accuracy: 0.9037  
- val\_loss: 0.5727 - val\_accuracy: 0.8104

Epoch 5/10

68/68 [=====] - 2s 22ms/step - loss: 0.2699 - accuracy: 0.9355  
- val\_loss: 0.5742 - val\_accuracy: 0.8139

Epoch 6/10

68/68 [=====] - 1s 21ms/step - loss: 0.1604 - accuracy: 0.9583  
- val\_loss: 0.7745 - val\_accuracy: 0.8124

Epoch 7/10

68/68 [=====] - 1s 22ms/step - loss: 0.1157 - accuracy: 0.9730  
- val\_loss: 0.8664 - val\_accuracy: 0.8127

Epoch 8/10

68/68 [=====] - 1s 21ms/step - loss: 0.0904 - accuracy: 0.9812  
- val\_loss: 1.0608 - val\_accuracy: 0.8115

Epoch 9/10

68/68 [=====] - 1s 22ms/step - loss: 0.0719 - accuracy: 0.9878  
- val\_loss: 1.2308 - val\_accuracy: 0.8082

Epoch 10/10

68/68 [=====] - 1s 22ms/step - loss: 0.0615 - accuracy: 0.9907  
- val\_loss: 1.2759 - val\_accuracy: 0.8036

179/179 [=====] - 1s 3ms/step

	precision	recall	f1-score	support
0	0.81	0.83	0.82	3035
1	0.80	0.78	0.79	2689
accuracy			0.80	5724
macro avg	0.80	0.80	0.80	5724
weighted avg	0.80	0.80	0.80	5724

## Trying out the model on current headlines

The Onion is not the only source of news satire nowadays and the Huffington Post is not the only source of news. Here I tested some headlines from various sarcastic and serious news networks.

It seems that this model is not very good at determining satire from other news websites. This should be improved by adding more headlines from different sites, thereby incorporating more writing styles into the model.

In [106...

```
serious_headlines = [  
    ["sudan fighting: army says foreign nationals to be evacuated"],#bbc  
    ["oklahoma man saves wife from being sucked away by tornado"],#bbc  
    ["nfl suspensions expose the league's problem with sports gambling"] #NYT  
]  
  
satire_headlines = [  
    ["not just for crimes: the 5 most polite acts ever committed in international water  
    ["9 names to consider before naming your child aiden"],#hard times  
    ["places you've left your water bottle, and whether you can get it back"] #the new  
]  
  
print(cnn_model.predict(serious_headlines))  
print(cnn_model.predict(satire_headlines))
```

```
1/1 [=====] - 0s 45ms/step  
[[-0.08028802]  
 [ 3.2397485 ]  
 [-0.05846712]]  
1/1 [=====] - 0s 48ms/step  
[[ 0.04883295]  
 [ 1.5674242 ]  
 [-0.23062049]]
```