Health and Wellness Text Classification

This notebook provides a comprehensive overview of building a deep learning algorithm for text classification to predict whether a news story is about health and wellness.We'll be using the k-train, and identifying precision and recall.

Model Training with ktrain We utilized the ktrain library for model development, experimenting with different deep learning architectures:

DistilBERT with ngram=1: This configuration represents embeddings for individual tokens.

DistilBERT with ngram=2: Here, embeddings represent both single tokens and two-word combinations.

BERT with ngram=2: Similar to the previous model but employing the larger BERT architecture.

Please note that this notebook was executed in Kaggle due to the requirement for a robust GPU infrastructure to handle BERT computations

Data

The dataset utilized in this project is the News Category Training Data sourced from HuffPost. It comprises approximately 200,000 news headlines spanning the years 2012 to 2018. Each entry in the dataset contains several attributes:

- **Category**: The category of the news article, covering various topics such as politics, entertainment, business, sports, health, and wellness.
- **Headline**: The headline or title of the news article, providing a brief summary of the content.
- **Authors**: The names of the authors or contributors who wrote the article.
- **Link**: The hyperlink to access the full article online.
- **Short Description**: A concise description or summary of the article content, providing additional context beyond the headline.
- **Date**: The date when the article was published or made available.

For this project, the focus is on articles related to health and wellness, which are identified based on specific categories such as "Healthy Living" and "Wellness." The dataset's size and variety make it suitable for training deep learning models for text classification and analysis.

Imports

```
!pip install ktrain
```

```
:a 0:00:0000:0100:01

l0/site-packages (from ktrain) (1.2.2)
ıon3.10/site-packages (from ktrain) (3.7.4)
.10/site-packages (from ktrain) (2.1.4)
ɔython3.10/site-packages (from ktrain) (1.0.3)
ıte-packages (from ktrain) (2.31.0)
e-packages (from ktrain) (1.3.2)
site-packages (from ktrain) (21.3)


:a 0:00:00

-packages (from ktrain) (0.42.1)
:hon3.10/site-packages (from ktrain) (3.3.2)









ɔython3.10/site-packages (from ktrain) (4.37.0)
.10/site-packages (from ktrain) (0.1.99)




:a 0:00:00
-packages (from keras_bert>=0.86.0->ktrain) (1.24.4)


ıs_bert>=0.86.0->ktrain)


eras_bert>=0.86.0->ktrain)


).40.0->keras_bert>=0.86.0->ktrain)


ɾmer==0.40.0->keras_bert>=0.86.0->ktrain)


ɾas_bert>=0.86.0->ktrain)
```

```
->keras-transformer==0.40.0->keras_bert>=0.86.0->ktrain)

on3.10/site-packages (from matplotlib>=3.0.0->ktrain) (1.2.0)
```

```
import ktrain
import pandas as pd
import numpy as np
```

GPU check

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Sat Feb  3 04:26:03 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.129.03             Driver Version: 535.129.03   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla P100-PCIE-16GB           Off | 00000000:00:04.0 Off |                    0 |
| N/A   32C    P0              27W / 250W |      0MiB / 16384MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Load the data

The data file is loaded from Kaggle datasets.

```
reviews = pd.read_json("../input/news-category/news_category_trainingdata.json")
```

Inspect the data

```
reviews.head()
```

	category	headline	authors	link	short
0	CRIME	There Were 2 Mass Shootings In Texas Last Week...	Melissa Jeltsen	https://www.huffingtonpost.com/entry/texas-ama...	hus

Prepare the data

Both headline and short description have meaninful data to parse. So they are combined.

```
reviews['combined_text'] = reviews['headline'] + ' ' + reviews['short_description']
```

Our initial step involves data preparation, where we aim to transform a categorical column into a binary indicator reflecting whether an article relates to health and wellness. Specifically, we assign a value of **1** to articles categorized under '**Healthy Living**' or '**Wellness**', indicating their relevance to health and wellness topics. And articles falling under other categories are assigned a value of **0**, signifying their lack of association with health and wellness. It's important to note that within our dataset, the categories '**Healthy Living**' and '**Wellness**' denote content related to health, while other categories represent diverse topics outside this domain."

```
reviews[(reviews['category'].str.contains("HEALTHY LIVING")) | (reviews['category'].str.cont
```

	category	headline	authors	
7578	HEALTHY LIVING	To The People Who Say 'I'm Tired' When Someone...	The Mighty, ContributorWe face disability, dis...	https://www.huffingtonpost.com/entry/to-the
7693	HEALTHY LIVING	Eating Shake Shack Made Me Feel Healthier Than...	Colleen Werner, ContributorCampus Editor-at-Large	https://www.huffingtonpost.com/entry/eating
7747	HEALTHY LIVING	How To Stay Updated On The News Without Losing...	Lindsay Holmes	https://www.huffingtonpost.com/entry/anxie
		27 Perfect		

```
reviews['healthy'] = np.where((reviews['category'] == 'HEALTHY LIVING') | (reviews['category
```

```
reviews['healthy'].describe()
```

```
count    200853.000000
mean          0.122084
std           0.327384
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           1.000000
Name: healthy, dtype: float64
```

Balance the data

To ensure a balanced dataset including all health and wellness articles, the sample_amount variable is set equal to the total number of articles falling within this category. This approach guarantees that the sample size aligns precisely with the count of health and wellness entries, thereby maintaining balance across the dataset.

```
sample_amount =  len(reviews[reviews["healthy"] == 1])
```

```
healthy = reviews[reviews['healthy'] == 1].sample(n=sample_amount)
not_healthy = reviews[reviews['healthy'] == 0].sample(n=sample_amount)


review_sample = pd.concat([healthy,not_healthy])


review_sample.describe()
```

	date	healthy
count	49042	49042.000000
mean	2014-08-06 00:22:50.645569280	0.500000
min	2012-01-28 00:00:00	0.000000
25%	2013-02-23 00:00:00	0.000000
50%	2014-03-07 00:00:00	0.500000
75%	2015-12-21 00:00:00	1.000000
max	2018-05-26 00:00:00	1.000000
std	NaN	0.500005

```
target_names = ['NOT HEALTH AND WELLNESS','HEALTH AND WELLNESS']
```

Model Training

We will try the following scenarios with ktrain

- distilbert with ngram=1
- distilbert with ngram=2
- bert with ngram=2

distilbert and ngram=1

A value of 1 signifies that the embeddings represent individual words or tokens extracted from the combined text. In other words, each embedding corresponds to a single word or token in the text data.

```
train, val, preprocess = ktrain.text.texts_from_df(
    review_sample,
```

```
    "combined_text",
    label_columns=["healthy"],
    val_df=None,
    max_features=20000,
    maxlen=512,
    val_pct=0.1,
    ngram_range=1,
    preprocess_mode="distilbert",
    verbose=1
)
```

```
['not_healthy', 'healthy']
        not_healthy  healthy
166475          0.0      1.0
153552          0.0      1.0
131008          0.0      1.0
34840           1.0      0.0
194221          0.0      1.0
['not_healthy', 'healthy']
        not_healthy  healthy
196307          1.0      0.0
99759           0.0      1.0
143251          1.0      0.0
124584          0.0      1.0
135429          0.0      1.0
config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
pytorch_model.bin:   0%|          | 0.00/268M [00:00<?, ?B/s]
/opt/conda/lib/python3.10/site-packages/torch/_utils.py:831: UserWarning: TypedStorage i
  return self.fget.__get__(instance, owner)()
preprocessing train...
language: en
train sequence lengths:
        mean : 32
        95percentile : 58
        99percentile : 68
tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
Is Multi-Label? False
preprocessing test...
language: en
test sequence lengths:
        mean : 32
        95percentile : 58
```

```
model = preprocess.get_classifier()
learner = ktrain.get_learner(model, train_data=train, val_data=val, batch_size=32)
```

Here we will determine the optimal learning rate for our training process.

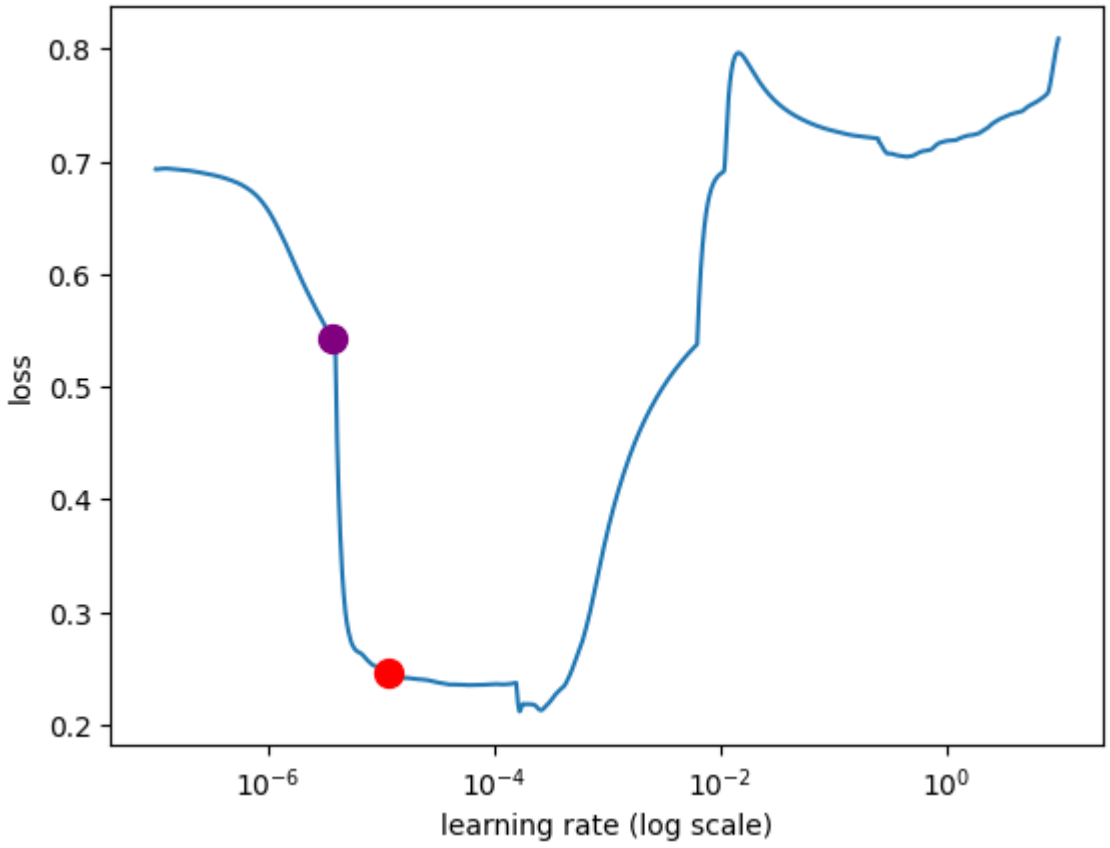
```
learner.lr_find(max_epochs=5)
```

```
simulating training for different learning rates... this may take a few moments...
Epoch 1/5
1379/1379 [==============================] - 1419s 1s/step - loss: 0.5265 - accuracy: 0
Epoch 2/5
1379/1379 [==============================] - 1408s 1s/step - loss: 0.2381 - accuracy: 0
Epoch 3/5
1379/1379 [==============================] - 1408s 1s/step - loss: 0.5435 - accuracy: 0
Epoch 4/5
1379/1379 [==============================] - 1405s 1s/step - loss: 0.7197 - accuracy: 0
Epoch 5/5
1379/1379 [==============================] - 1402s 1s/step - loss: 0.8301 - accuracy: 0


done.
Please invoke the Learner.lr_plot() method to visually inspect the loss plot to help ide
```

```
learner.lr_plot(suggest=True)
```

```
Three possible suggestions for LR from plot:
        Longest valley (red): 1.19E-05
        Min numerical gradient (purple): 3.66E-06
        Min loss divided by 10 (omitted from plot): 1.69E-05
```



Three strategies can be employed to select the learning rate. One approach involves identifying the 'Longest valley' (Red), indicating the point where the loss begins to diverge. Alternatively, the 'Steepest part' (Purple) of the loss curve can be used, representing the starting point of a valley where the loss consistently decreases. For this experiment, the third strategy was adopted, which involves finding the minimum loss and dividing it by 10 to set a maximum threshold for the model's loss.

Used the third option for this experiment

```
lr_est = learner.lr_estimate()
lr_est
```

```
     (3.6646402e-06, 1.6943589434958996e-05, 1.1872607e-05)
```

Actual Training

```
history=learner.autofit(
    1.7e-5,
    checkpoint_folder='checkpoint',
    epochs=10,
    early_stopping=True
)
```

```
     begin training using triangular learning rate policy with max lr of 1.7e-05...
     Epoch 1/10
     1380/1380 [==============================] - 1450s 1s/step - loss: 0.2829 - accuracy: 0
     Epoch 2/10
     1380/1380 [==============================] - 1421s 1s/step - loss: 0.1753 - accuracy: 0
     Epoch 3/10
     1380/1380 [==============================] - ETA: 0s - loss: 0.1328 - accuracy: 0.9526Re
     1380/1380 [==============================] - 1421s 1s/step - loss: 0.1328 - accuracy: 0
     Epoch 3: early stopping
     Weights from best epoch have been loaded into model.
```

```
predictor = ktrain.get_predictor(learner.model, preproc=preprocess)
```

Saved the predictor so I could reload later if this was the best model

```
predictor.save("health_wellness")
```

```
!zip -r health_wellness.zip health_wellness
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has alrea
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
  adding: health_wellness/ (stored 0%)
  adding: health_wellness/config.json (deflated 44%)
  adding: health_wellness/vocab.txt (deflated 53%)
  adding: health_wellness/tf_model.preproc (deflated 47%)
  adding: health_wellness/tf_model.h5 (deflated 8%)
  adding: health_wellness/tokenizer.json (deflated 71%)
  adding: health_wellness/tokenizer_config.json (deflated 76%)
  adding: health_wellness/special_tokens_map.json (deflated 42%)
```

Validation Results

```
validation = learner.validate(val_data=val, print_report=True)
```

```
154/154 [==============================] - 22s 131ms/step
              precision    recall  f1-score   support

           0       0.93      0.90      0.92      2424
           1       0.91      0.94      0.92      2481

    accuracy                           0.92      4905
   macro avg       0.92      0.92      0.92      4905
weighted avg       0.92      0.92      0.92      4905
```

It is evident that the precision and recall metrics for health and wellness articles are already at 91% and 94%, respectively. This performance surpasses the precision and recall thresholds set as baseline in the Final project.

distilbert with ngram=2

ngram=2 option includes both single words and two word combinations as list of embeddings in Training and Validation. The hypothesis is that this combination could improve the performance of prediction

```
train, val, preprocess = ktrain.text.texts_from_df(
    review_sample,
    "combined_text",
    label_columns=["healthy"],
```

```
    val_df=None,
    max_features=20000,
    maxlen=512,
    val_pct=0.1,
    ngram_range=2,
    preprocess_mode="distilbert",
    verbose=1
)
```

```
     ['not_healthy', 'healthy']
             not_healthy  healthy
     146234          0.0      1.0
     682             1.0      0.0
     157582          0.0      1.0
     67475           0.0      1.0
     149572          0.0      1.0
     ['not_healthy', 'healthy']
             not_healthy  healthy
     137291          1.0      0.0
     81882           0.0      1.0
     187551          1.0      0.0
     101473          0.0      1.0
     120042          1.0      0.0
     config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
     model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
     preprocessing train...
     language: en
     train sequence lengths:
             mean : 32
             95percentile : 58
             99percentile : 68
     tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
     vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
     tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
     Is Multi-Label? False
     preprocessing test...
     language: en
     test sequence lengths:
             mean : 32
             95percentile : 59
             99percentile : 68
```

```
model = preprocess.get_classifier()
learner = ktrain.get_learner(model, train_data=train, val_data=val, batch_size=32)
```

```
learner.lr_find(max_epochs=5)
```

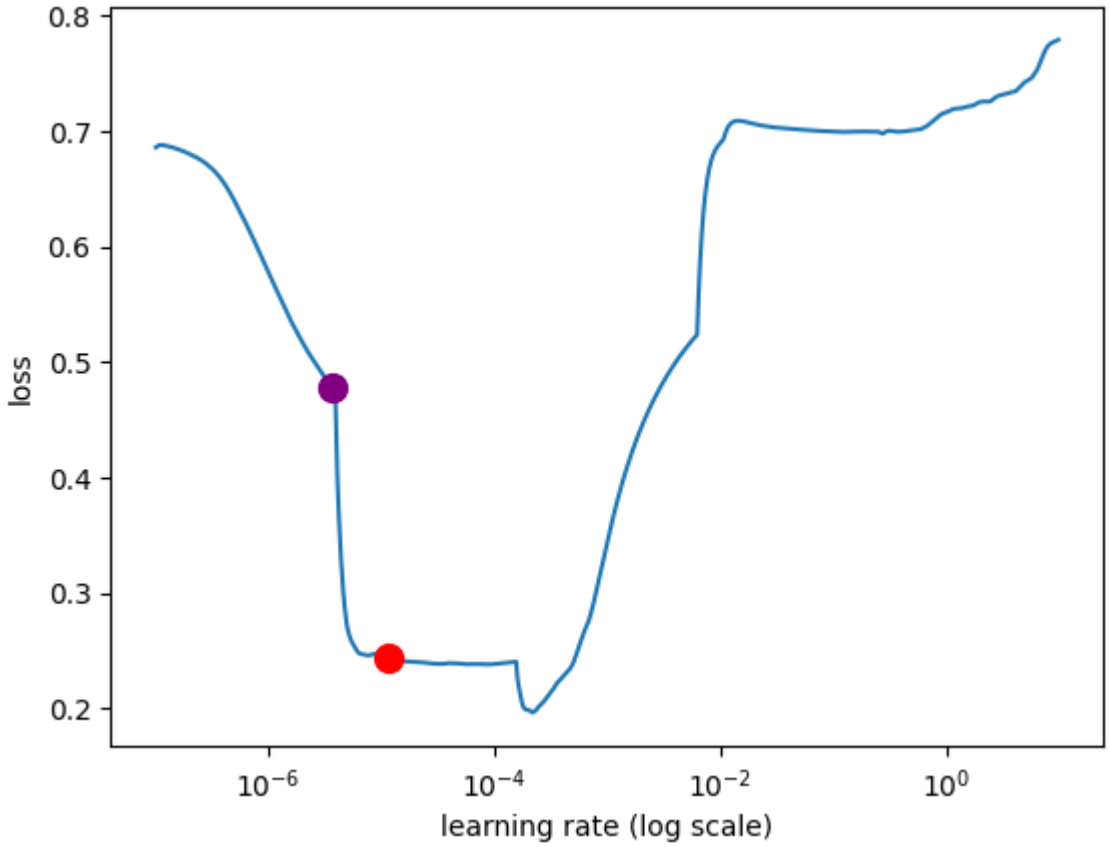
```
     simulating training for different learning rates... this may take a few moments...
     Epoch 1/5
     1379/1379 [==============================] - 1417s 1s/step - loss: 0.4660 - accuracy: 0
     Epoch 2/5
     1379/1379 [==============================] - 1408s 1s/step - loss: 0.2411 - accuracy: 0
     Epoch 3/5
```

```
1379/1379 [==============================] - 1407s 1s/step - loss: 0.5301 - accuracy: 0
Epoch 4/5
1379/1379 [==============================] - 1404s 1s/step - loss: 0.6995 - accuracy: 0
Epoch 5/5
1379/1379 [==============================] - 1401s 1s/step - loss: 0.7873 - accuracy: 0


done.
Please invoke the Learner.lr_plot() method to visually inspect the loss plot to help ide
```

```
learner.lr_plot(suggest=True)
```

```
Three possible suggestions for LR from plot:
        Longest valley (red): 1.18E-05
        Min numerical gradient (purple): 3.66E-06
        Min loss divided by 10 (omitted from plot): 2.21E-05
```




```
lr_est = learner.lr_estimate()
lr_est
```

```
(3.6646402e-06, 2.213254920206964e-05, 1.184093e-05)
```

```
history=learner.autofit(
    2.2e-5,
    checkpoint_folder='checkpoint',
```

```
    epochs=10,
    early_stopping=True
)
```

```
begin training using triangular learning rate policy with max lr of 2.2e-05...
Epoch 1/10
1380/1380 [==============================] - 1448s 1s/step - loss: 0.2726 - accuracy: 0
Epoch 2/10
1380/1380 [==============================] - 1419s 1s/step - loss: 0.1711 - accuracy: 0
Epoch 3/10
1380/1380 [==============================] - ETA: 0s - loss: 0.1221 - accuracy: 0.9569Re
1380/1380 [==============================] - 1419s 1s/step - loss: 0.1221 - accuracy: 0
Epoch 3: early stopping
Weights from best epoch have been loaded into model.
```

```
predictor = ktrain.get_predictor(learner.model, preproc=preprocess)
```

```
validation = learner.validate(val_data=val, print_report=True)
```

```
154/154 [==============================] - 21s 125ms/step
              precision    recall  f1-score   support

           0       0.94      0.92      0.93      2419
           1       0.92      0.95      0.93      2486

    accuracy                           0.93      4905
   macro avg       0.93      0.93      0.93      4905
weighted avg       0.93      0.93      0.93      4905
```

From the observations above, it's apparent that there was a slight improvement in the performance of predictions. With this model configuration, the precision and recall metrics stand at 92% and 95%, respectively

```
predictor.save("health_wellness_2")
```

```
!zip -r health_wellness_2.zip health_wellness_2
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has alrea
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
  adding: health_wellness_2/ (stored 0%)
  adding: health_wellness_2/tf_model.preproc (deflated 47%)
  adding: health_wellness_2/config.json (deflated 44%)
```

```
  adding: health_wellness_2/special_tokens_map.json (deflated 42%)
  adding: health_wellness_2/tf_model.h5 (deflated 8%)
  adding: health_wellness_2/vocab.txt (deflated 53%)
  adding: health_wellness_2/tokenizer.json (deflated 71%)
  adding: health_wellness_2/tokenizer_config.json (deflated 76%)
```

bert with ngram=2

```
train, val, preprocess = ktrain.text.texts_from_df(
    review_sample,
    "combined_text",
    label_columns=["healthy"],
    val_df=None,
    max_features=20000,
    maxlen=512,
    val_pct=0.1,
    ngram_range=2,
    preprocess_mode="bert",
    verbose=1
)
```

```
['not healthy', 'healthy']
```

Here we have used a batch size of 6 due to memory limitations, as larger batch sizes caused memory issues. However, this constraint could potentially impact the performance of the model.

```
model = ktrain.text.text_classifier('bert', train, preproc=preprocess)

learner = ktrain.get_learner(model, train_data=train, val_data=val, batch_size=6)
```

```
Is Multi-Label? False
maxlen is 512
/opt/conda/lib/python3.10/site-packages/keras/src/initializers/initializers.py:120: User
  warnings.warn(
done.
```

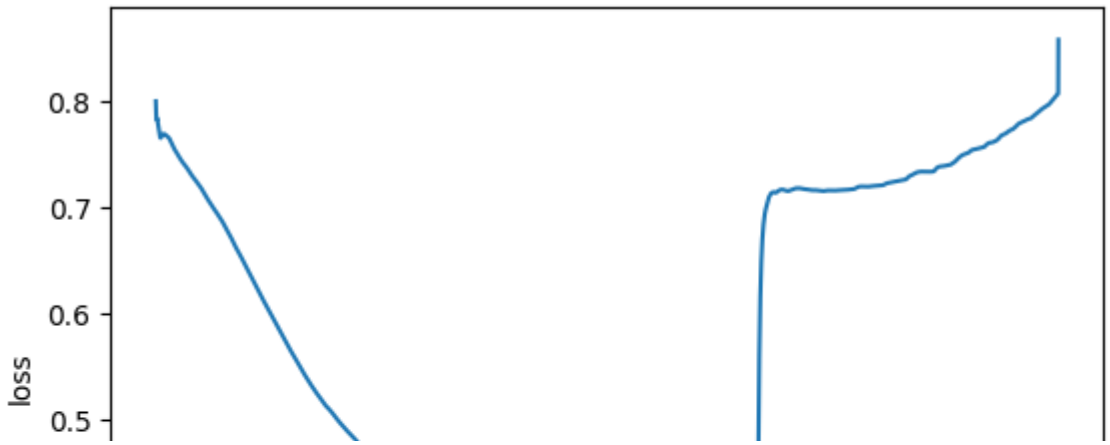
```
learner.lr_find(max_epochs=5)
```

```
simulating training for different learning rates... this may take a few moments...
/opt/conda/lib/python3.10/site-packages/keras/src/engine/training.py:3103: UserWarning:
  saving_api.save_model(
Epoch 1/5
7357/7357 [==============================] - 3627s 490ms/step - loss: 0.4012 - accuracy
Epoch 2/5
7357/7357 [==============================] - 3602s 490ms/step - loss: 0.2866 - accuracy
Epoch 3/5
7357/7357 [==============================] - 3603s 490ms/step - loss: 0.8079 - accuracy
Epoch 4/5
7357/7357 [==============================] - 5s 667us/step - loss: 1.1898 - accuracy: 0


done.
Please invoke the Learner.lr_plot() method to visually inspect the loss plot to help ide
```

```
learner.lr_plot(suggest=True)
```

```
Three possible suggestions for LR from plot:
        Longest valley (red): 2.45E-06
        Min numerical gradient (purple): 3.92E-06
        Min loss divided by 10 (omitted from plot): 2.05E-06
```




```
lr_est = learner.lr_estimate()
lr_est
```

```
(3.919755e-06, 2.0451505406526847e-06, 2.4505473e-06)
```

```
history=learner.autofit(
    2e-6,
    checkpoint_folder='checkpoint',
    epochs=10,
    early_stopping=True
)
```

```
begin training using triangular learning rate policy with max lr of 2e-06...
Epoch 1/10
7357/7357 [==============================] - 3755s 508ms/step - loss: 0.3316 - accuracy
Epoch 2/10
7357/7357 [==============================] - 3728s 507ms/step - loss: 0.2108 - accuracy
Epoch 3/10
7357/7357 [==============================] - 3728s 507ms/step - loss: 0.1781 - accuracy
Epoch 4/10
7357/7357 [==============================] - 3728s 507ms/step - loss: 0.1531 - accuracy
Epoch 5/10
7357/7357 [==============================] - ETA: 0s - loss: 0.1302 - accuracy: 0.9520Re
7357/7357 [==============================] - 3728s 507ms/step - loss: 0.1302 - accuracy
Epoch 5: early stopping
Weights from best epoch have been loaded into model.
```

```
predictor = ktrain.get_predictor(learner.model, preproc=preprocess)
```

```
validation = learner.validate(val_data=val, print_report=True)
```

```
154/154 [==============================] - 116s 730ms/step
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      2482
           1       0.93      0.93      0.93      2423

    accuracy                           0.93      4905
   macro avg       0.93      0.93      0.93      4905
weighted avg       0.93      0.93      0.93      4905
```

Based on the results above, we observe that the precision and recall for '**Health and Wellness**' articles are 93%. This performance is comparable to the DistilBERT model, which achieved precision and recall scores of 92% and 95%, respectively. Given the simplicity of architecture and comparable performance, I have decided to proceed with DistilBERT with ngram=2 for the remainder of the project.

```
predictor.save("health_wellness_3")
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/engine/training.py:3103: UserWarning:
  saving_api.save_model(
```

```
!zip -r health_wellness_3.zip health_wellness_3
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has alrea
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
  adding: health_wellness_3/ (stored 0%)
  adding: health_wellness_3/tf_model.preproc (deflated 48%)
  adding: health_wellness_3/tf_model.h5 (deflated 12%)
```

Inspecting the drivers of prediction

Here we load the best model that was trained - distilbert with ngram=2

```
predictor = ktrain.load_predictor('../input/saved-model/health_wellness_2')
```

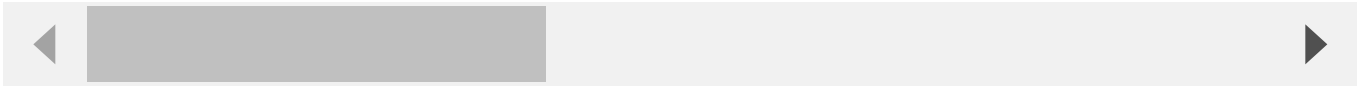
```
!pip3 install -q git+https://github.com/amaiya/eli5@tfkeras_0_10_1
```

We will test the model using the same set of documents provided in the class.

```
test_docs = [
'Stress May Be Your Heart’s Worst Enemy Psychological stress activates the fear center in th
'Exercising to Slim Down? Try Getting Bigger. It’s high time for women to reclaim the real s
'What Are Your Food Resolutions for the New Year? Join us for the Eat Well Challenge startir
'Why We All Need to Have More Fun. Prioritizing fun may feel impossible right now. But this
'Cuomo Will Not Be Prosecuted in Groping Case, Albany D.A. Says. The district attorney descr
'A Film Captures Jewish Life in a Polish Town Before the Nazis Arrived. A documentary based
             ]
```

```
for i, text in enumerate(test_docs):
  probs = predictor.predict(text, return_proba=True)
  print("---------------------------")
  print('The probability this is healthy is %s' % probs[1])
  print(text)
```

```
---------------------------
The probability this is healthy is 0.99463415
Stress May Be Your Heart’s Worst Enemy Psychological stress activates the fear center ir
---------------------------
The probability this is healthy is 0.98836446
Exercising to Slim Down? Try Getting Bigger. It’s high time for women to reclaim the rea
---------------------------
The probability this is healthy is 0.92587644
What Are Your Food Resolutions for the New Year? Join us for the Eat Well Challenge star
---------------------------
The probability this is healthy is 0.99004793
Why We All Need to Have More Fun. Prioritizing fun may feel impossible right now. But th
---------------------------
The probability this is healthy is 0.0034020075
Cuomo Will Not Be Prosecuted in Groping Case, Albany D.A. Says. The district attorney de
---------------------------
The probability this is healthy is 0.0031613447
A Film Captures Jewish Life in a Polish Town Before the Nazis Arrived. A documentary bas
```



It's evident that the model performs exceptionally well on these examples.

Conculsion

Through experimentation and evaluation, we found that all model variants performed reasonably well for the task of health and wellness text classification. However, we selected the DistilBERT model with ngram=2 as our final choice due to its simplicity and competitive performance.

Moving forward, further optimization and refinement of the chosen model could lead to even better performance. Additionally, exploring cloud-based solutions with more powerful hardware may enable us to experiment with larger models and batch sizes, potentially improving performance further.

Overall, this project demonstrates the effectiveness of deep learning for text classification tasks in the health and wellness domain and highlights the importance of model evaluation, interpretation, and continuous improvement.