```
# based on classify_text_with_bert program
# https://www.tensorflow.org/text/tutorials/classify_text_with_bert
# Eric G. Suchanek, PhD
# (c)2022 Eric G. Suchanek, all rights reserved
```

Classify text with BERT

In this notebook, you will:

- Load the IMDB dataset
- Load a BERT model from TensorFlow Hub
- Build your own model by combining BERT with a classifier
- Train your own model, fine-tuning BERT as part of that
- Save your model and use it to classify sentences

If you're new to working with the IMDB dataset, please see Basic text classification for more details.

```
# library imports
import re
import os
import shutil
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')

from keras.preprocessing.text import text_to_word_sequence

!pip install -q tensorflow==2.8.*
!pip install -q tf-models-official==2.8.*
!pip install -q tensorflow_hub==2.8.*
!pip install -q tensorflow_text==2.8.*

!pip install scikit-multilearn
from skmultilearn.model_selection import iterative_train_test_split

```

```
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
from official.nlp import optimization  # to create AdamW optimizer


#!pip install keras

#from official.nlp import optimization  # to create AdamW optimizer

tf.get_logger().setLevel('ERROR')

```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
ERROR: Could not find a version that satisfies the requirement tensorflow_hub==2.8.* (from versions: 0.1.0, 0
ERROR: No matching distribution found for tensorflow_hub==2.8.*
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-multilearn in /usr/local/lib/python3.7/dist-packages (0.2.0)
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

NotebookDir = "/content/drive/My Drive/Colab Notebooks"
DataDir = "/content/drive/My Drive/data"
CleanDir = "/content/drive/My Drive/data/clean"
ImgDir = "/content/drive/My Drive/img"
LogDir = "/contents/drive/My Drive/logs"
ModelDir = "/contents/drive/My Drive/models"
ModulelDir = "/content/drive/My Drive/Colab Notebooks/bby"

# install the bby module
os.chdir(ModulelDir)
!pip install .

# set our device appropriately
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('No gpu')
else:
    print(gpu_info)

# note that mps device is available on M1 Mac hardware if properly installed
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

```

```
Mounted at /content/drive
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Processing /content/drive/My Drive/Colab Notebooks/bby
  DEPRECATION: A future pip version will change local packages to be built in-place without first copying to
   pip 21.3 will remove support for this functionality. You can find discussion regarding this at https://git
Building wheels for collected packages: BBY
  Building wheel for BBY (setup.py) ... done
  Created wheel for BBY: filename=BBY-0.4-py3-none-any.whl size=7295 sha256=458d117b2b7c6040b863ef80b0aac3d72
  Stored in directory: /tmp/pip-ephem-wheel-cache-p8lkmbkf/wheels/ca/db/c0/53f76aec514218649c05e0de324fbde98d
Successfully built BBY
Installing collected packages: BBY
  Attempting uninstall: BBY
    Found existing installation: BBY 0.4
    Uninstalling BBY-0.4:
      Successfully uninstalled BBY-0.4
Successfully installed BBY-0.4
Sat Jul 16 04:43:20 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| 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...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   35C    P0    28W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
Num GPUs Available:  1
```

```
#import bby
#from bby.util import clean_doc, tb_enrich, nps_cleanstring
from bby.util import detokenize

```

```
# Now get the comment into a form suitable for tokenizing
def sent_to_words(sentences):
    for sentence in sentences:
        yield(text_to_word_sequence(sentence))
    return

def str_it(_ls):
    ls = str(_ls)
    word_tokens = word_tokenize(ls)
    ls = [w for w in word_tokens]

    ls = " ".join(ls)
    return ls

def write_txt_files(sentencelist, prefix='nps'):
    lc = 1
    print(f'   writing: {len(sentencelist)} files...')
    for item in sentencelist:
        outfilename = f'{prefix}_{lc}.txt'
        try:
            outfile = open(outfilename, 'w')
        except OSError as error:
            print(f'Cannot create file: {outfilename}. Failed with error: {error}! Exiting')
            return
        outfile.write(item)
        outfile.write('\n')
        lc += 1
    outfile.close()
    return

def iterative_train_test_split_dataframe(X, y, test_size):
    df_index = np.expand_dims(X.index.to_numpy(), axis=1)
    X_train, y_train, X_test, y_test = iterative_train_test_split(df_index, y, test_size = test_size)
    X_train = X.loc[X_train[:,0]]
    X_test = X.loc[X_test[:,0]]
    return X_train, y_train, X_test, y_test
```

```
# create directories for train/test split of nps data in the format needed by
# keras read_text_dataset
#
nltk.download('punkt')
```

```

def write_NPS_text_dataset(input_filename, rootpath='npsdata', frac=0.2):
    curr_dir = os.getcwd()
    os.chdir(DataDir)

  # remove the entire tree first
    try:
      print('Removing prior train/test directories...')
      shutil.rmtree(rootpath)
    except OSError as error:
        print(error)

    # make the train and test subdirs for trainpath and testpath
    trainpath = os.path.join(rootpath,'train')
    testpath = os.path.join(rootpath,'test')
    # print(f'train: {trainpath}, test: {testpath}')

    try:
        os.makedirs(trainpath)
    except OSError as error:
        print(error)
        return

    try:
        os.makedirs(testpath)
    except OSError as error:
        print(error)
        return

    class_list = ['detractor', 'passive', 'promoter']

    # try to read the input file
    try:
        os.path.exists(input_filename)
    except OSError as error:
        print(error)
        print(f'Cant read input file {input_filename}. Fatal, exiting')
        return

    NPS_df = pd.read_csv(input_filename, index_col='respid2')


    # now make the class subdirectories for training and testing
    os.chdir(DataDir)
    os.chdir(trainpath)
    for cls in class_list:
        os.mkdir(cls)

    os.chdir(DataDir)
    os.chdir(testpath)
    for cls in class_list:
        os.mkdir(cls)

    prom_list_mask = NPS_df['NPS_Code'] == 2
    pass_list_mask = NPS_df['NPS_Code'] == 1
    det_list_mask = NPS_df['NPS_Code'] == 0

    # nps_list = NPS_df['NPSCommentCleaned'].apply(str_it)

    prom_list = NPS_df[prom_list_mask]
    pass_list = NPS_df[pass_list_mask]
    det_list = NPS_df[det_list_mask]

    prom_list_len = prom_list.shape[0]
    pass_list_len = pass_list.shape[0]
    det_list_len = det_list.shape[0]

    len_list = [prom_list_len, pass_list_len, det_list_len]
    print (f'Overall Distribution: Promoters: {prom_list_len}, Passives: {pass_list_len}, Detractors: {det_list_len}')

    shortest = np.argmin(len_list)
    sample_size = int(np.round(len_list[shortest] * frac))
    print(sample_size)

    prom_sample_size = int(np.round(prom_list_len * frac))
    pass_sample_size = int(np.round(pass_list_len * frac))
    det_sample_size = int(np.round(det_list_len * frac))

    # these subsets represent the test subset
    prom_list_test = prom_list.sample(sample_size)
    pass_list_test = pass_list.sample(sample_size)
    det_list_test = det_list.sample(sample_size)

    prom_list_train = prom_list[~prom_list.apply(tuple,1).isin(prom_list_test.apply(tuple, 1))]
    pass_list_train = pass_list[~pass_list.apply(tuple,1).isin(pass_list_test.apply(tuple, 1))]
    det_list_train = det_list[~det_list.apply(tuple,1).isin(det_list_test.apply(tuple, 1))]

    prom_train_sent = prom_list_train['NPSCommentCleaned'].apply(str_it)
    prom_test_sent = prom_list_test['NPSCommentCleaned'].apply(str_it)

    pass_train_sent = pass_list_train['NPSCommentCleaned'].apply(str_it)
    pass_test_sent = pass_list_test['NPSCommentCleaned'].apply(str_it)

    det_train_sent = det_list_train['NPSCommentCleaned'].apply(str_it)
    det_test_sent = det_list_test['NPSCommentCleaned'].apply(str_it)

    print(f'Promoters:')
    print(f'Training size: {prom_list_train.shape[0]}')
```

```
print(f'Testing size {prom_list_test.shape[0]}')
print(f'Original size: {prom_list.shape[0]}')

print(f'\nPassives:')
print(f'Training size: {pass_list_train.shape[0]}')
print(f'Testing size {pass_list_test.shape[0]}')
print(f'Original size: {pass_list.shape[0]}')

print(f'\nDetractors:')
print(f'Training size: {det_list_train.shape[0]}')
print(f'Testing size {det_list_test.shape[0]}')
print(f'Original size: {det_list.shape[0]}')

# Checking balance of target classes after equalization
sentiments = list(NPS_df["NPS® Breakdown"].unique())
sentiment_nums = [len(NPS_df[NPS_df["NPS® Breakdown"] == sentiment]) / len(NPS_df) for sentiment in sentiments]

print (f'\nOverall Distribution: Promoters: {prom_list_len}, Passives: {pass_list_len}, Detractors: {det_list_len}')
plt.bar(sentiments, sentiment_nums)

# now write the training files by class
os.chdir(DataDir)
os.chdir(trainpath)
print(f'Writing training files...')
os.chdir('promoter')
write_txt_files(prom_train_sent)
os.chdir('..')

#os.chdir('passive')
#write_txt_files(pass_train_sent)
#os.chdir('..')
os.chdir('detractor')
write_txt_files(det_train_sent)

# now write testing files by class
os.chdir(DataDir)
os.chdir(testpath)
print(f'\nWriting testing files...')

os.chdir('promoter')
write_txt_files(prom_test_sent)
#os.chdir('..')
#os.chdir('passive')
#write_txt_files(pass_test_sent)
os.chdir('..')
os.chdir('detractor')
write_txt_files(det_test_sent)

os.chdir(curr_dir)
```

```
    return

```

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

```
# print(f'current: {os.getcwd()}')
# os.listdir()
```

```
os.chdir(DataDir)

os.listdir()
#write_NPS_text_dataset('NPS_NATL_subset.csv')
```

```
['aclImdb',
 'NPS_NATL_archive.xlsx',
 'npsdata',
 'npsdata_bert',
 'model.png',
 'NPS_NATL_subset.csv',
 'NPS_Natl_cleaned.csv']
```

```
#
os.chdir(DataDir)

NPS_df = pd.read_csv('NPS_NATL_subset.csv', index_col='respid2', usecols=['respid2', 'NPS_Code', 'NPSCommentCleaned'])
y = NPS_df['NPS_Code'].values.tolist()
y = np.array(y)
labels = tf.keras.utils.to_categorical(y, 3, dtype="float32")
del y

np.random.seed(42)
X_train_df, y_train_df, X_test_df, y_test_df = iterative_train_test_split_dataframe(NPS_df, labels, 0.2)


#NPS_df.head(5)

```

```
AUTOTUNE = tf.data.AUTOTUNE
batch_size2 = 32
seed = 42
curr_dir = os.getcwd()
class_list = ['detractor', 'passive', 'promoter']
```

```

os.chdir(DataDir)

raw_train_ds = tf.keras.utils.text_dataset_from_directory(
    'npsdata/train',
    batch_size=batch_size2,
    validation_split=0.2,
    subset='training',
    seed=seed)

class_names = raw_train_ds.class_names
train_ds = raw_train_ds.cache().prefetch(buffer_size=AUTOTUNE)

val_ds = tf.keras.utils.text_dataset_from_directory(
    'npsdata/train',
    batch_size=batch_size2,
    validation_split=0.2,
    subset='validation',
    seed=seed)

val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

test_ds = tf.keras.utils.text_dataset_from_directory(
    'npsdata/test',
    batch_size=batch_size2)

test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
Found 15801 files belonging to 2 classes.
Using 12641 files for training.
Found 15801 files belonging to 2 classes.
Using 3160 files for validation.
Found 3114 files belonging to 2 classes.
```

```
for text_batch, label_batch in train_ds.take(1):
  for i in range(3):
    print(f'Comment: {text_batch.numpy()[i]}')
    label = label_batch.numpy()[i]
    print(f'Label : {label} ({class_names[label]})')
```

```
Comment: b'awesome service\n'
Label : 1 (promoter)
Comment: b'got all the advice needed\n'
Label : 1 (promoter)
Comment: b'initial representative bryce misled and misguided me lied to me not acceptable\n'
Label : 0 (detractor)
```

Choose a BERT model to fine-tune

bert_model_name: small_bert/bert_en_uncased_L-4_H-512_A-8 ▼

Show code

```
BERT model selected           : https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1
Preprocess model auto-selected: https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3
```

```
bert_preprocess_model = hub.KerasLayer(tfhub_handle_preprocess)
```

```
text_test = ['this is such an amazing movie!']
text_preprocessed = bert_preprocess_model(text_test)

print(f'Keys       : {list(text_preprocessed.keys())}')
print(f'Shape      : {text_preprocessed["input_word_ids"].shape}')
print(f'Word Ids   : {text_preprocessed["input_word_ids"][0, :12]}')
print(f'Input Mask : {text_preprocessed["input_mask"][0, :12]}')
print(f'Type Ids   : {text_preprocessed["input_type_ids"][0, :12]}')
```

```
Keys       : ['input_mask', 'input_word_ids', 'input_type_ids']
Shape      : (1, 128)
Word Ids   : [ 101 2023 2003 2107 2019 6429 3185  999  102    0    0    0]
Input Mask : [1 1 1 1 1 1 1 1 1 0 0 0]
Type Ids   : [0 0 0 0 0 0 0 0 0 0 0 0]
```

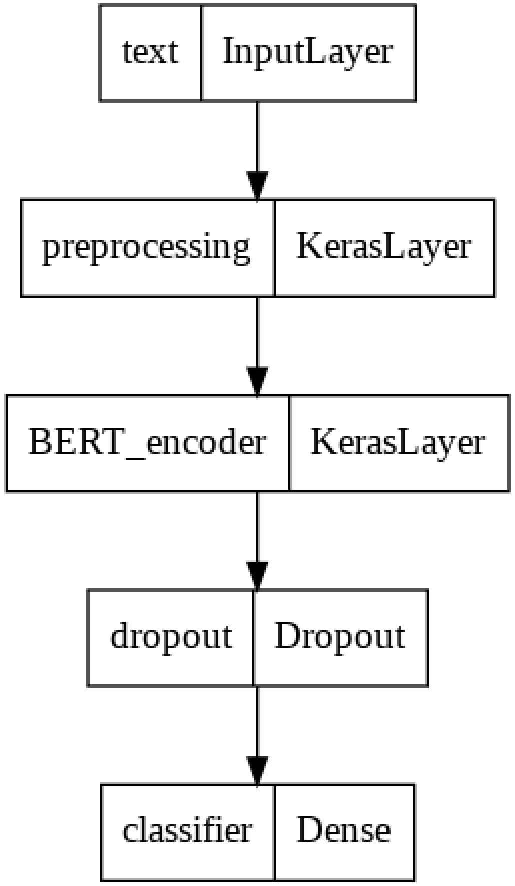
```
bert_model = hub.KerasLayer(tfhub_handle_encoder)
```

```
def build_classifier_model():
  text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
  preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
  encoder_inputs = preprocessing_layer(text_input)
  encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
  outputs = encoder(encoder_inputs)
  net = outputs['pooled_output']
  net = tf.keras.layers.Dropout(0.1)(net)
  #net = tf.keras.layers.Dense(3, activation='relu', name='relu')(net)
  net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)
  return tf.keras.Model(text_input, net)
```

```
classifier_model = build_classifier_model()
bert_raw_result = classifier_model(tf.constant(text_test))
print(tf.sigmoid(bert_raw_result))
```

```
tf.Tensor([[0.17450172]], shape=(1, 1), dtype=float32)
```

```
tf.keras.utils.plot_model(classifier_model)
```



```
loss = tf.keras.losses.BinaryCrossentropy(from_logits=False)
metrics = tf.metrics.BinaryAccuracy()
```

```
from official.nlp import optimization  # to create AdamW optimizer

epochs = 20
steps_per_epoch = tf.data.experimental.cardinality(train_ds).numpy()
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
optimizer = optimization.create_optimizer(init_lr=init_lr,
                                          num_train_steps=num_train_steps,
                                          num_warmup_steps=num_warmup_steps,
                                          optimizer_type='adamw')
```

```
classifier_model.compile(optimizer=optimizer,
                         loss=loss,
                         metrics=metrics)
```

```
print(f'Training model with {tfhub_handle_encoder}')
history = classifier_model.fit(x=train_ds,
                               validation_data=val_ds,
                               epochs=epochs)
```

```
Training model with https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1
Epoch 1/20
396/396 [==============================] - 580s 1s/step - loss: 2.0316 - binary_accuracy: 0.7950 - val_loss:
Epoch 2/20
396/396 [==============================] - 64s 162ms/step - loss: 0.9838 - binary_accuracy: 0.9146 - val_loss
Epoch 3/20
396/396 [==============================] - 64s 162ms/step - loss: 0.8959 - binary_accuracy: 0.9289 - val_loss
Epoch 4/20
396/396 [==============================] - 64s 162ms/step - loss: 0.8372 - binary_accuracy: 0.9375 - val_loss
Epoch 5/20
396/396 [==============================] - 64s 162ms/step - loss: 0.7398 - binary_accuracy: 0.9457 - val_loss
Epoch 6/20
396/396 [==============================] - 64s 162ms/step - loss: 0.6818 - binary_accuracy: 0.9509 - val_loss
Epoch 7/20
396/396 [==============================] - 64s 162ms/step - loss: 0.6422 - binary_accuracy: 0.9536 - val_loss
Epoch 8/20
396/396 [==============================] - 64s 161ms/step - loss: 0.6115 - binary_accuracy: 0.9558 - val_loss
Epoch 9/20
```

```
396/396 [==============================] - 64s 162ms/step - loss: 0.5688 - binary_accuracy: 0.9592 - val_loss
Epoch 10/20
396/396 [==============================] - 64s 162ms/step - loss: 0.5626 - binary_accuracy: 0.9604 - val_loss
Epoch 11/20
396/396 [==============================] - 64s 162ms/step - loss: 0.5149 - binary_accuracy: 0.9642 - val_loss
Epoch 12/20
396/396 [==============================] - 64s 161ms/step - loss: 0.4850 - binary_accuracy: 0.9660 - val_loss
Epoch 13/20
396/396 [==============================] - 64s 162ms/step - loss: 0.4915 - binary_accuracy: 0.9661 - val_loss
Epoch 14/20
396/396 [==============================] - 64s 162ms/step - loss: 0.4817 - binary_accuracy: 0.9665 - val_loss
Epoch 15/20
396/396 [==============================] - 64s 161ms/step - loss: 0.4480 - binary_accuracy: 0.9693 - val_loss
Epoch 16/20
396/396 [==============================] - 64s 162ms/step - loss: 0.4301 - binary_accuracy: 0.9703 - val_loss
Epoch 17/20
396/396 [==============================] - 64s 162ms/step - loss: 0.4129 - binary_accuracy: 0.9719 - val_loss
Epoch 18/20
396/396 [==============================] - 64s 163ms/step - loss: 0.4186 - binary_accuracy: 0.9714 - val_loss
Epoch 19/20
396/396 [==============================] - 64s 162ms/step - loss: 0.4086 - binary_accuracy: 0.9723 - val_loss
Epoch 20/20
396/396 [==============================] - 64s 163ms/step - loss: 0.4041 - binary_accuracy: 0.9723 - val_loss
```

```python
loss, accuracy = classifier_model.evaluate(test_ds)

print(f'Loss: {loss}')
print(f'Accuracy: {accuracy}')
```

```
98/98 [==============================] - 177s 2s/step - loss: 0.7209 - binary_accuracy: 0.9512
Loss: 0.7209467887878418
Accuracy: 0.9511882066726685
```

```python
history_dict = history.history
print(history_dict.keys())

acc = history_dict['binary_accuracy']
val_acc = history_dict['val_binary_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

```

```
epochs = range(1, len(acc) + 1)
fig = plt.figure(figsize=(10, 6))
fig.tight_layout()

plt.subplot(2, 1, 1)
# r is for "solid red line"
plt.plot(epochs, loss, 'r', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
# plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
<matplotlib.legend.Legend at 0x7fd32475b850>
```

```
dataset_name = 'npsdata'
saved_model_path = './{}_bert'.format(dataset_name.replace('/', '_'))

classifier_model.save(saved_model_path, include_optimizer=False)
```

```
WARNING:absl:Found untraced functions such as restored_function_body, restored_function_body, restored_functi
```

```
reloaded_model = tf.saved_model.load(saved_model_path)
```

```
def print_my_examples(inputs, results):
  result_for_printing = \
    [f'input: {inputs[i]:<30} : score: {results[i][0]:.6f}'
                         for i in range(len(inputs))]
  print(*result_for_printing, sep='\n')
  print()


examples = [
    'eric is a wonderful agent!',  # this is the same sentence tried earlier
    'the service was good but the results were bad',
    'The geek squad is really good. The agent was super helpful',
    'The agent and service was slow and okish.',
    'The agent was terrible...'
]

reloaded_results = tf.sigmoid(reloaded_model(tf.constant(examples)))
original_results = tf.sigmoid(classifier_model(tf.constant(examples)))

print('Results from the saved model:')
print_my_examples(examples, reloaded_results)
print('Results from the model in memory:')
print_my_examples(examples, original_results)
```

```
Results from the saved model:
input: eric is a wonderful agent!     : score: 0.998978
input: the service was good but the results were bad : score: 0.005050
input: The geek squad is really good. The agent was super helpful : score: 0.999527
input: The agent and service was slow and okish. : score: 0.007649
input: The agent was terrible...      : score: 0.006861
```

```
Results from the model in memory:
input: eric is a wonderful agent!     : score: 0.998978
input: the service was good but the results were bad : score: 0.005050
input: The geek squad is really good. The agent was super helpful : score: 0.999527
input: The agent and service was slow and okish. : score: 0.007649
input: The agent was terrible...      : score: 0.006861
```

```
serving_results = reloaded_model \
            .signatures['serving_default'](tf.constant(examples))

serving_results = tf.sigmoid(serving_results['classifier'])

print_my_examples(examples, serving_results)
```

```
input: eric is a wonderful agent!     : score: 0.998978
input: the service was good but the results were bad : score: 0.005050
input: The geek squad is really good. The agent was super helpful : score: 0.999527
input: The agent and service was slow and okish. : score: 0.007649
input: The agent was terrible...      : score: 0.006861
```

● ×