CODE: Sentiment Classification - Impact of Pretrained Word Embedding

```
import numpy as np

import tensorflow_datasets as tfds
import tensorflow as tf

data, info = tfds.load('imdb_reviews', with_info=True, as_supervised=True)
train, test = data['train'], data['test']
```

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (download: 80.23
Dl Completed...: 100%        1/1 [00:05<00:00, 5.32s/ url]

Dl Size...: 100%        80/80 [00:05<00:00, 23.93 MiB/s]


Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
100%                                              24999/25000 [00:00<00:00, 116024.35 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
100%                                              24999/25000 [00:00<00:00, 118347.06 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
100%                                              49999/50000 [00:00<00:00, 133471.40 examples/s]
WARNING:absl:Dataset is using deprecated text encoder API which will be removed
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_r
```

```
VOCAB_SIZE = 10000
BATCH_SIZE = 64
embedding_dim=100
max_length = 120
trunc_type= 'post'
UNK="<UNK>"
```

```
## Processing the data

train_input = []  #X_train
test_input = []   #X_test
train_labels = [] #y_train
test_labels = []  #y_test

for i,j in train:
    train_input.append(str(i.numpy()))
    train_labels.append(j.numpy())

for i,j in test:
    test_input.append(str(i.numpy()))
    test_labels.append(j.numpy())


train_labels = np.array(train_labels)
test_labels = np.array(test_labels)
```

```
print(test_input[1])
```

```
e they concern his moral integrity and we are never quite sure whether it remains
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words = VOCAB_SIZE, oov_token=UNK)
tokenizer.fit_on_texts(train_input)
word_index = tokenizer.word_index
```

```
seq = tokenizer.texts_to_sequences(train_input)
padding = pad_sequences(seq, maxlen=max_length, truncating = trunc_type)
test_seq = tokenizer.texts_to_sequences(test_input)
test_padded = pad_sequences(test_seq, maxlen=max_length)
```

```
print("train : ", len(seq[0]))
print("test : ", len(test_seq[0]))
```

```
train :  118
test :  173
```

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

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

```
import string
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

output = []

```

```
for line in train_input:
  tokens = word_tokenize(line)
  tokens = [word.lower() for word in tokens]
  processed = str.maketrans('','', string.punctuation)
  translated = [word.translate(processed) for word in tokens]
  words = [word for word in translated if word.isalpha()]
  stopWords = set(stopwords.words('english'))
  words = [word for word in words if not word in stopWords]
  output.append(words)
```

```
print(output[0])
print(len(output))
```

```
['b', 'absolutely', 'terrible', 'movie', 'nt', 'lured', 'christopher', 'walken',
25000
```

```
  import gensim

  model_1 = gensim.models.Word2Vec(sentences = output, size= embedding_dim, window
  words = list(model_1.wv.vocab)
```

```
model_1.wv.save_word2vec_format("w2v_embedding.txt", binary = False)
```

```
import os
path_to_w2v_file = "w2v_embedding.txt"

embeddings_index = {}
with open(path_to_w2v_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 93966 word vectors.
```

```
num_tokens = len(word_index) + 1
hits = 0
misses = 0

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:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
```

```
Converted 70670 words (15869 misses)
```

```
from keras.layers import Embedding
from keras.initializers import Constant

embedding_layer = Embedding(num_tokens, embedding_dim, embeddings_initializer= C
```

```
## LSTM Model
from keras.models import Sequential
from keras.layers import Dense, LSTM, GRU
from keras.layers.embeddings import Embedding

model_1 = Sequential()
model_1.add(embedding_layer)
model_1.add(LSTM(units=64, dropout=0.2))
model_1.add(Dense(32, activation='relu'))
model_1.add(Dense(1, activation='sigmoid'))

model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 120, 100)          8654000

 lstm (LSTM)                 (None, 64)                42240

 dense (Dense)               (None, 32)                2080

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 8,698,353
Trainable params: 44,353
Non-trainable params: 8,654,000
_________________________________________________________________
```

```
model_1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
history = model_1.fit(padding, train_labels, epochs=20, validation_data = (test_p
```

```
Epoch 1/20
782/782 [==============================] - 37s 38ms/step - loss: 0.5588 - accura
Epoch 2/20
```

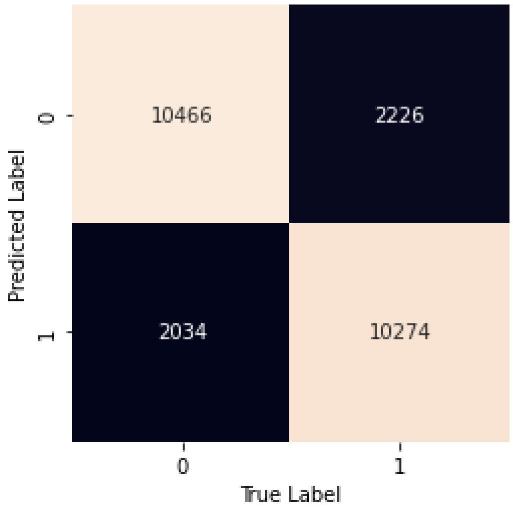
```
782/782 [==============================] - 29s 37ms/step - loss: 0.4431 - accura
Epoch 3/20
782/782 [==============================] - 29s 37ms/step - loss: 0.4215 - accura
Epoch 4/20
782/782 [==============================] - 29s 37ms/step - loss: 0.4035 - accura
Epoch 5/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3999 - accura
Epoch 6/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3898 - accura
Epoch 7/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3851 - accura
Epoch 8/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3772 - accura
Epoch 9/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3726 - accura
Epoch 10/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3632 - accura
Epoch 11/20
782/782 [==============================] - 31s 40ms/step - loss: 0.3573 - accura
Epoch 12/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3478 - accura
Epoch 13/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3412 - accura
Epoch 14/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3316 - accura
Epoch 15/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3266 - accura
Epoch 16/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3170 - accura
Epoch 17/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3071 - accura
Epoch 18/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3018 - accura
Epoch 19/20
782/782 [==============================] - 29s 37ms/step - loss: 0.2941 - accura
Epoch 20/20
782/782 [==============================] - 28s 36ms/step - loss: 0.2828 - accura
```

```python
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score

predicted_1 = model_1.predict(test_padded)

pred_1 = np.zeros(len(predicted_1))
for i, score in enumerate(predicted_1):
    if score > 0.5:
      pred_1[i] = 1
    else:
      pred_1[i] = 0

print(metrics.classification_report(test_labels, pred_1))
print("Precision = ",precision_score(test_labels, pred_1))
print("Recall = ", recall_score(test_labels, pred_1))
```

```
              precision    recall  f1-score   support

           0       0.82      0.84      0.83     12500
           1       0.83      0.82      0.83     12500

    accuracy                           0.83     25000
   macro avg       0.83      0.83      0.83     25000
weighted avg       0.83      0.83      0.83     25000

Precision =  0.8347416314592135
Recall =  0.82192
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

mat = confusion_matrix(test_labels, pred_1)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True Label")
plt.ylabel("Predicted Label")
```

```
Text(91.68, 0.5, 'Predicted Label')
```



GLOVE

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2022-03-07 05:16:00--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2022-03-07 05:16:00--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connecte
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
```

```
--2022-03-07 05:16:00--  http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip        100%[===================>] 822.24M  5.11MB/s    in 2m 40s

2022-03-07 05:18:41 (5.13 MB/s) - 'glove.6B.zip' saved [862182613/862182613]
```

```
import os
path_to_glove_file = "glove.6B.100d.txt"

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

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

```
num_tokens = len(word_index) + 1
embedding_dim = 100 ## 100 dimensions
hits = 0 ## number of words that were found in the pretrained model
misses = 0 ## number of words that were missing in the pretrained model

# Prepare embedding matrix for our word list
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:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
```

```
Converted 60197 words (26342 misses)
```

```
from keras.layers import Embedding
from keras.initializers import Constant

embedding_layer = Embedding(num_tokens, embedding_dim, embeddings_initializer= Cc
## LSTM Model
from keras.models import Sequential
from keras.layers import Dense, LSTM, GRU
from keras.layers.embeddings import Embedding

model_2 = Sequential()
model_2.add(embedding_layer)
model_2.add(LSTM(units=64, dropout=0.2))
model_2.add(Dense(32, activation='relu'))
model_2.add(Dense(1, activation='sigmoid'))

model_2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 120, 100)          8654000

 lstm_2 (LSTM)               (None, 64)                42240

 dense_4 (Dense)             (None, 32)                2080

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 8,698,353
Trainable params: 44,353
Non-trainable params: 8,654,000
_________________________________________________________________
```

```
model_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
history = model_2.fit(padding, train_labels, epochs=20, validation_data = (test_p
```

```
Epoch 1/20
782/782 [==============================] - 31s 37ms/step - loss: 0.6761 - accura
Epoch 2/20
782/782 [==============================] - 28s 36ms/step - loss: 0.4945 - accura
Epoch 3/20
782/782 [==============================] - 28s 36ms/step - loss: 0.4385 - accura
Epoch 4/20
782/782 [==============================] - 28s 36ms/step - loss: 0.4172 - accura
Epoch 5/20
782/782 [==============================] - 28s 36ms/step - loss: 0.4015 - accura
Epoch 6/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3898 - accura
Epoch 7/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3760 - accura
Epoch 8/20
782/782 [==============================] - 29s 37ms/step - loss: 0.3621 - accura
Epoch 9/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3513 - accura
Epoch 10/20
```

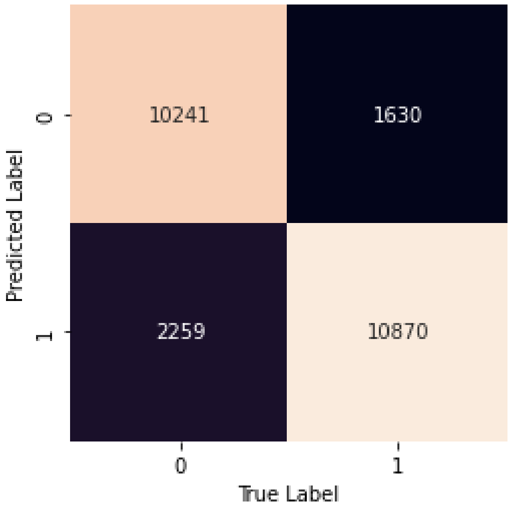
```
782/782 [==============================] - 28s 36ms/step - loss: 0.3393 - accura
Epoch 11/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3271 - accura
Epoch 12/20
782/782 [==============================] - 28s 36ms/step - loss: 0.3163 - accura
Epoch 13/20
782/782 [==============================] - 29s 37ms/step - loss: 0.3029 - accura
Epoch 14/20
782/782 [==============================] - 28s 36ms/step - loss: 0.2969 - accura
Epoch 15/20
782/782 [==============================] - 29s 36ms/step - loss: 0.2831 - accura
Epoch 16/20
782/782 [==============================] - 29s 37ms/step - loss: 0.2760 - accura
Epoch 17/20
782/782 [==============================] - 28s 36ms/step - loss: 0.2648 - accura
Epoch 18/20
782/782 [==============================] - 28s 36ms/step - loss: 0.2575 - accura
Epoch 19/20
782/782 [==============================] - 28s 36ms/step - loss: 0.2511 - accura
Epoch 20/20
782/782 [==============================] - 28s 36ms/step - loss: 0.2413 - accura
```

```python
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score

predicted_2 = model_2.predict(test_padded)

pred_2 = np.zeros(len(predicted_2))
for i, score in enumerate(predicted_2):
    if score > 0.5:
      pred_2[i] = 1
    else:
      pred_2[i] = 0

print(metrics.classification_report(test_labels, pred_2))
print("Precision = ",precision_score(test_labels, pred_2))
print("Recall = ", recall_score(test_labels, pred_2))
```

```
              precision    recall  f1-score   support

           0       0.86      0.82      0.84     12500
           1       0.83      0.87      0.85     12500

    accuracy                           0.84     25000
   macro avg       0.85      0.84      0.84     25000
weighted avg       0.85      0.84      0.84     25000

Precision =  0.8279381521821921
Recall =  0.8696
```

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

mat = confusion_matrix(test_labels, pred_2)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True Label")
plt.ylabel("Predicted Label")
```

```
Text(91.68, 0.5, 'Predicted Label')
```



0s completed at 7:50 AM