```
fsentence=[
          "I really like this book",
          "I love this place"
]
```

```
import tensorflow
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
```

```
one_hot_rep= [one_hot(words,100) for words in sentence]
```

```
one_hot_rep
```

```
[[73, 81, 86, 62, 78], [73, 12, 62, 76]]
```

```
length= 8
embedded_doc= pad_sequences(one_hot_rep, padding='pre', maxlen= length)
print(embedded_doc)
```

```
[[ 0  0  0 73 81 86 62 78]
 [ 0  0  0  0 73 12 62 76]]
```

```
dim=10
vocab_size=100
model= Sequential()
model.add(Embedding(vocab_size ,dim, input_length= length))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
embedding (Embedding)        (None, 8, 10)             1000
=================================================================
Total params: 1,000
Trainable params: 1,000
Non-trainable params: 0
_________________________________________________________________
```

```
pred= model.predict(embedded_doc)
```

```
pred
```

```
array([[[-0.04287918, -0.02877076, -0.02354172, -0.03075363,
         -0.0178406 , -0.03285166, -0.01275345, -0.01005472,
         -0.00189047, -0.02027624],
        [-0.04287918, -0.02877076, -0.02354172, -0.03075363,
         -0.0178406 , -0.03285166, -0.01275345, -0.01005472,
         -0.00189047, -0.02027624],
        [-0.04287918, -0.02877076, -0.02354172, -0.03075363,
         -0.0178406 , -0.03285166, -0.01275345, -0.01005472,
         -0.00189047, -0.02027624],
        [ 0.02780923,  0.01220452, -0.03800594,  0.04233992,
         -0.00432057,  0.03962095, -0.04240117, -0.03157319,
          0.01541325,  0.03793525],
        [-0.04040948, -0.00493157, -0.02408328, -0.03020506,
         -0.04969352,  0.00653899, -0.03759919, -0.00504839,
          0.0105625 , -0.04016125],
        [ 0.00627334, -0.02896903,  0.03973602, -0.0031049 ,
         -0.00641809,  0.01902397,  0.01207932,  0.04797149,
         -0.04365454,  0.02577943],
        [-0.01120736,  0.00095668, -0.02445908,  0.02159306,
          0.01325509, -0.03709564,  0.01575788, -0.02221253,
          0.04730154,  0.00085545],
        [ 0.00779371,  0.00252414,  0.0146768 ,  0.00528085,
         -0.02239087,  0.01802284, -0.01720225,  0.01783724,
         -0.01662058,  0.03386276]],

       [[-0.04287918, -0.02877076, -0.02354172, -0.03075363,
         -0.0178406 , -0.03285166, -0.01275345, -0.01005472,
         -0.00189047, -0.02027624],
        [-0.04287918, -0.02877076, -0.02354172, -0.03075363,
```

```
        -0.0178406 , -0.03285166, -0.01275345, -0.01005472,
        -0.00189047, -0.02027624],
       [-0.04287918, -0.02877076, -0.02354172, -0.03075363,
        -0.0178406 , -0.03285166, -0.01275345, -0.01005472,
        -0.00189047, -0.02027624],
       [-0.04287918, -0.02877076, -0.02354172, -0.03075363,
        -0.0178406 , -0.03285166, -0.01275345, -0.01005472,
        -0.00189047, -0.02027624],
       [ 0.02780923,  0.01220452, -0.03800594,  0.04233992,
        -0.00432057,  0.03962095, -0.04240117, -0.03157319,
         0.01541325,  0.03793525],
       [-0.0103317 ,  0.02695252,  0.02376086, -0.03908849,
        -0.01476322,  0.03212133,  0.0121193 , -0.03322773,
        -0.04861431, -0.03230001],
       [-0.01120736,  0.00095668, -0.02445908,  0.02159306,
         0.01325509, -0.03709564,  0.01575788, -0.02221253,
         0.04730154,  0.00085545],
       [-0.04504723,  0.01991005,  0.01779404,  0.04675931,
         0.02806688, -0.01103956,  0.03985474, -0.04167704,
        -0.0275653 ,  0.02900727]]], dtype=float32)
```

```
pred.shape
```

```
(2, 8, 10)
```

```
pred[0][0] # sentence 1
```

```
array([-0.04287918, -0.02877076, -0.02354172, -0.03075363, -0.0178406 ,
       -0.03285166, -0.01275345, -0.01005472, -0.00189047, -0.02027624],
      dtype=float32)
```

```
pred[0][1] # sentence 2
```

```
array([-0.04287918, -0.02877076, -0.02354172, -0.03075363, -0.0178406 ,
       -0.03285166, -0.01275345, -0.01005472, -0.00189047, -0.02027624],
      dtype=float32)
```

CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
cv= CountVectorizer()

bow= cv.fit_transform(sentence)
print(bow)
```

```
  (0, 4)	1
  (0, 1)	1
  (0, 5)	1
  (0, 0)	1
  (1, 5)	1
  (1, 2)	1
  (1, 3)	1
```

```
feature_names= cv.get_feature_names()
```

```
print(feature_names)
```

```
['book', 'like', 'love', 'place', 'really', 'this']
```

```
import pandas as pd
pd.DataFrame(bow.toarray(), columns= feature_names)
```

	book	like	love	place	really	this
0	1	1	0	0	1	1
1	0	0	1	1	0	1

TFIDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
tv= TfidfVectorizer()
```

```
tv_vector= tv.fit_transform(sentence)
print(tv_vector)
```

```
  (0, 0)	0.534046329052269
  (0, 5)	0.37997836159100784
  (0, 1)	0.534046329052269
  (0, 4)	0.534046329052269
  (1, 3)	0.6316672017376245
  (1, 2)	0.6316672017376245
  (1, 5)	0.4494364165239821
```

```
feature_names= tv.get_feature_names()
```

```
import pandas as pd
pd.DataFrame(tv_vector.toarray(), columns= feature_names)
```

	book	like	love	place	really	this
0	0.534046	0.534046	0.000000	0.000000	0.534046	0.379978
1	0.000000	0.000000	0.631667	0.631667	0.000000	0.449436

BBC News data

Multiclass classification using Word2Vec and LSTM

```
import pandas as pd
data=pd.read_csv('/content/drive/MyDrive/bbc_news_mixed (1).csv')
```

```
data.head()
```

	text	label
0	Cairn shares slump on oil setback\n\nShares in...	business
1	Egypt to sell off state-owned bank\n\nThe Egyp...	business
2	Cairn shares up on new oil find\n\nShares in C...	business

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import LabelBinarizer
data.label.value_counts()
```

```
sport            511
business         510
politics         417
tech             401
entertainment    386
Name: label, dtype: int64
```

```
label2= pd.get_dummies(data["label"])
```

```
label2.head()
```

	business	entertainment	politics	sport	tech
0	1	0	0	0	0
1	1	0	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	1	0	0	0	0

```
y = LabelBinarizer().fit_transform(label2)
```

```
y[:5]
```

```
array([[1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0]])
```

```
z= pd.DataFrame(y)
z.value_counts()
```

```
0  1  2  3  4
0  0  0  1  0    511
1  0  0  0  0    510
0  0  1  0  0    417
      0  0  1    401
   1  0  0  0    386
dtype: int64
```

```
data.head()
```

	text	label
0	Cairn shares slump on oil setback\n\nShares in...	business
1	Egypt to sell off state-owned bank\n\nThe Egyp...	business
2	Cairn shares up on new oil find\n\nShares in C...	business
3	Low-cost airlines hit Eurotunnel\n\nChannel Tu...	business
4	Parmalat to return to stockmarket\n\nParmalat,...	business

```
from gensim.utils import simple_preprocess
preprocessed_bbc = data.text.apply(lambda x: simple_preprocess(x))
preprocessed_bbc.head()
```

```
0    [cairn, shares, slump, on, oil, setback, share...
1    [egypt, to, sell, off, state, owned, bank, the...
2    [cairn, shares, up, on, new, oil, find, shares...
3    [low, cost, airlines, hit, eurotunnel, channel...
```

```
4    [parmalat, to, return, to, stockmarket, parmal...
Name: text, dtype: object
```

```
# import word2vec
from gensim.models import Word2Vec

# train a word2vec model from the given data set
w2v_model = Word2Vec(preprocessed_bbc, size=300, min_count=2, sg=1)
```

```
print('vocabulary size:', len(w2v_model.wv.vocab))
```

```
vocabulary size: 18588
```

```
w2v_model.wv.most_similar('oil')
```

```
[('gas', 0.8538216352462769),
 ('costs', 0.8051249980926514),
 ('giant', 0.8016246557235718),
 ('energy', 0.797722339630127),
 ('fuel', 0.7918999195098877),
 ('telecoms', 0.7897387742996216),
 ('unit', 0.7854053974151611),
 ('exports', 0.7794791460037231),
 ('china', 0.7793854475021362),
 ('steel', 0.7775402069091797)]
```

```
def get_embedding_w2v(doc_tokens, pre_trained):
    embeddings = []
    if pre_trained:
        model = w2vec
    else:
        model = w2v_model
    for tok in doc_tokens:
        if tok in model.wv.vocab:
            embeddings.append(model.wv.word_vec(tok))
    return np.mean(embeddings, axis=0)
```

```
import numpy as np
```

```
X_w2v_model = preprocessed_bbc.apply(lambda x: get_embedding_w2v(x, pre_trained=0))
X_w2v_model = pd.DataFrame(X_w2v_model.tolist())
print('X shape:', X_w2v_model.shape)
```

```
X shape: (2225, 300)
```

```
from sklearn.model_selection import train_test_split
X_train_wm, X_test_wm, y_train_wm, y_test_wm = train_test_split(X_w2v_model, y)
```

```
X_train_wm.shape,X_test_wm.shape
```

```
((1668, 300), (557, 300))
```

```
y_train_wm.shape, y_test_wm.shape
```

```
((1668, 5), (557, 5))
```

```
X_train_wm=np.array(X_train_wm).reshape(1668, 300,1)
X_train_wm.shape
```

```
(1668, 300, 1)
```

```
X_test_wm=np.array(X_test_wm).reshape(557, 300,1)
X_test_wm.shape
```

```
(557, 300, 1)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dense

model1=Sequential()
model1.add(LSTM(100,input_shape=(300,1)))
model1.add(Dense(8,activation="relu"))
model1.add(Dense(5,activation="softmax"))

model1.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
```

```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100)               40800
_________________________________________________________________
dense (Dense)                (None, 8)                 808
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 45
=================================================================
Total params: 41,653
Trainable params: 41,653
Non-trainable params: 0
_________________________________________________________________
```

```
model1.fit(X_train_wm, y_train_wm,validation_data=(X_test_wm, y_test_wm), epochs=100)
```

```
Epoch 63/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3504 - accuracy: 0.8765 - val_loss: 0.3100 - val_acc
Epoch 64/100
53/53 [==============================] - 1s 22ms/step - loss: 0.3675 - accuracy: 0.8717 - val_loss: 0.3668 - val_acc
Epoch 65/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3252 - accuracy: 0.8801 - val_loss: 0.2983 - val_acc
Epoch 66/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3492 - accuracy: 0.8687 - val_loss: 0.3380 - val_acc
Epoch 67/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3420 - accuracy: 0.8765 - val_loss: 0.3729 - val_acc
Epoch 68/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3328 - accuracy: 0.8807 - val_loss: 0.2970 - val_acc
Epoch 69/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3396 - accuracy: 0.8723 - val_loss: 0.2946 - val_acc
Epoch 70/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3545 - accuracy: 0.8699 - val_loss: 0.3640 - val_acc
Epoch 71/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3289 - accuracy: 0.8777 - val_loss: 0.3077 - val_acc
Epoch 72/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3580 - accuracy: 0.8645 - val_loss: 0.3476 - val_acc
Epoch 73/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3233 - accuracy: 0.8825 - val_loss: 0.3573 - val_acc
```

```
Epoch 74/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3147 - accuracy: 0.8885 - val_loss: 0.3204 - val_acc
Epoch 75/100
53/53 [==============================] - 1s 22ms/step - loss: 0.3334 - accuracy: 0.8819 - val_loss: 0.4168 - val_acc
Epoch 76/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3549 - accuracy: 0.8759 - val_loss: 0.3675 - val_acc
Epoch 77/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3308 - accuracy: 0.8867 - val_loss: 0.2927 - val_acc
Epoch 78/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3419 - accuracy: 0.8747 - val_loss: 0.2977 - val_acc
Epoch 79/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3377 - accuracy: 0.8807 - val_loss: 0.3274 - val_acc
Epoch 80/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3090 - accuracy: 0.8891 - val_loss: 0.2752 - val_acc
Epoch 81/100
53/53 [==============================] - 1s 22ms/step - loss: 0.2936 - accuracy: 0.8915 - val_loss: 0.2746 - val_acc
Epoch 82/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3372 - accuracy: 0.8783 - val_loss: 0.2690 - val_acc
Epoch 83/100
53/53 [==============================] - 1s 20ms/step - loss: 0.3468 - accuracy: 0.8735 - val_loss: 0.2910 - val_acc
Epoch 84/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3369 - accuracy: 0.8789 - val_loss: 0.3336 - val_acc
Epoch 85/100
53/53 [==============================] - 1s 22ms/step - loss: 0.3126 - accuracy: 0.8855 - val_loss: 0.2739 - val_acc
Epoch 86/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3533 - accuracy: 0.8789 - val_loss: 0.3078 - val_acc
Epoch 87/100
53/53 [==============================] - 1s 22ms/step - loss: 0.3067 - accuracy: 0.8873 - val_loss: 0.3094 - val_acc
Epoch 88/100
53/53 [==============================] - 1s 22ms/step - loss: 0.3543 - accuracy: 0.8747 - val_loss: 0.2983 - val_acc
Epoch 89/100
53/53 [==============================] - 1s 22ms/step - loss: 0.3245 - accuracy: 0.8849 - val_loss: 0.2708 - val_acc
Epoch 90/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3280 - accuracy: 0.8753 - val_loss: 0.3132 - val_acc
Epoch 91/100
53/53 [==============================] - 1s 21ms/step - loss: 0.3130 - accuracy: 0.8891 - val_loss: 0.2929 - val_acc
```

```
model1.evaluate(X_test_wm,y_test_wm)
```

```
18/18 [==============================] - 0s 14ms/step - loss: 0.2550 - accuracy: 0.9013
[0.255048006772995, 0.9012567400932312]
```

```
classes=["business",  "entertainment",  "politics", "sport",  "tech"]
```

```
def prediction(doc):
  doc= simple_preprocess(doc)
  doc=get_embedding_w2v(doc, pre_trained=0)
  doc1=doc.reshape(1,300,1)
  p= model1.predict(doc1)
  return classes[np.argmax(p)]
```

```
doc1= "Pankaj Tripathi, currently seen on Criminal Justice: Behind Closed Doors, opens up about dealing with fame"
print(f"The article belongs to {prediction(doc1)} category",)
```

```
The article belongs to business category
```

```
doc2= "OnePlus 9 Alleged Live Images Tip Flat Hole-Punch Display, Reverse Wireless Charging Support"
print(f"The article belongs to {prediction(doc2)} category",)
```

```
The article belongs to tech category
```

```
doc3= "Tesla public company duties are a much bigger factor, but going private is impossible now (sigh),” Musk said in respo
print(f"The article belongs to {prediction(doc3)} category",)
```

```
The article belongs to tech category
```

```
doc4= "PSG sack Tuchel, Pochettino set to become new manager - reports."
print(f"The article belongs to {prediction(doc4)} category",)
```

```
The article belongs to sport category
```

```
doc5= "In a press conference on Tuesday, Kejriwal said the development of Uttar Pradesh has been held back by 'corrupt' lead
print(f"The article belongs to {prediction(doc5)} category",)
```

```
The article belongs to sport category
```

```
doc6= "RIL plans to rebrand the IMG Reliance as its completely owned subsidiary post-acquisition of 50 per cent shares held 
print(f"The article belongs to {prediction(doc6)} category",)
```

```
The article belongs to business category
```