Classificação_Multiclasses_de_Textos

May 23, 2021

1 Classificação multiclasses

Baseado em F. Chollet (2018).

Diferentemente das classificações que fizemos até aqui, sempre binárias (positivo/negativo; spam/ham...), existem classificações de dados em um número n de classes (ou categorias).

Neste exercício, usaremos um conjunto de dados de notícias da agência Reuters para uma classificação de assuntos. Há 46 categorias de assuntos possíveis, todas elas mutuamente excludentes, isto é, para cada notícia, existe somente uma categoria correta.

O corpus traz 8982 notícias para treinamento e outras 2246 para teste. Cada categoria tem pelo menos 10 ocorrências no treinamento.

#Carregamento dos dados e das etiquetas

[1]:
```
from keras.datasets import reuters

(dados_treino, etiquetas_treino), (dados_teste, etiquetas_teste) = reuters.
↪load_data(num_words=10000)  # Carregar só as 10.000 palavras mais frequentes
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/reuters.npz
2113536/2110848 [==============================] - 0s 0us/step
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/datasets/reuters.py:148:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/datasets/reuters.py:149:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

[2]:
```
### Conferindo...

print(len(dados_treino), len(dados_teste))
print(dados_treino[0])
```

```
8982 2246
[1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 3095, 111, 16, 369, 186, 90, 67,
7, 89, 5, 19, 102, 6, 19, 124, 15, 90, 67, 84, 22, 482, 26, 7, 48, 4, 49, 8,
864, 39, 209, 154, 6, 151, 6, 83, 11, 15, 22, 155, 11, 15, 7, 48, 9, 4579, 1005,
504, 6, 258, 6, 272, 11, 15, 22, 134, 44, 11, 15, 16, 8, 197, 1245, 90, 67, 52,
29, 209, 30, 32, 132, 6, 109, 15, 17, 12]
```

[3]:
```
### Só para conferir... ###

# Cria um dicionário com a forma {palavra: índice}
indice_de_palavra = reuters.get_word_index()
print('O índice de "after" é: ', indice_de_palavra['after'])

# Cria um dicionário que permite mapear uma palavra em função de um valor␣
 ↪numérico (seu índice)
palavra_de_indice = dict([(value, key) for (key, value) in indice_de_palavra.
 ↪items()])
print('A palavra de índice 100 é: ', palavra_de_indice[100])

# Gera uma string com o vetor das palavras decodificadas em uma dada resenha (0,␣
 ↪no caso)
# Os três primeiros índices são reservados nesse dataset ("padding", "início de␣
 ↪sequência" e "desconhecido")
# Lembre-se: o método .get() substitui chaves desconhecidas por um valor default␣
 ↪('<?>', no caso)
noticia_decodificada = ' '.join([palavra_de_indice.get(i-3, '<?>') for i in␣
 ↪dados_treino[0]])
print(noticia_decodificada)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/reuters_word_index.json
557056/550378 [==============================] - 0s 0us/step
O índice de "after" é:  89
A palavra de índice 100 é:  group
<?> <?> <?> said as a result of its december acquisition of space co it expects
earnings per share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986
the company said pretax net should rise to nine to 10 mln dlrs from six mln dlrs
in 1986 and rental operation revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it
said cash flow per share this year should be 2 50 to three dlrs reuter 3
```

[4]:
```
# As etiquetas são números inteiros no intervalo [0..45]
print(etiquetas_treino[0])
```

```
print(min(etiquetas_treino), max(etiquetas_treino))
```

```
3
0 45
```

2 Codificação em binários

Codificação (encoding) dos vetores de inteiros em matrizes binárias de índices de termos em documentos.

[5]:
```
import numpy as np

def binarizar(matriz_int, dim=10000):
    binarizado = np.zeros((len(matriz_int), dim))

    for e, vetor in enumerate(matriz_int):
        binarizado[e, vetor] = 1.

    return binarizado

# Conversão em binários dos testos das resenhas (variável X)
treino_x = binarizar(dados_treino)
teste_x = binarizar(dados_teste)
```

[6]:
```
treino_x[0]  # Só para conferir...
```

[6]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

[7]:
```
# Vetorização das etiquetas de assuntos [0..45]. Variável Y.

treino_y = binarizar(etiquetas_treino, dim=46)  # Repare o número de dimensões
 ↪(46)
teste_y = binarizar(etiquetas_teste, dim=46)
```

[8]:
```
print(len(treino_x), len(treino_y), len(teste_x), len(teste_y))  # Conferindo...
```

```
8982 8982 2246 2246
```

[9]:
```
treino_y[0]  # Conferindo...
```

[9]:
```
array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

3 Criação do modelo de rede

Partição dos dados de treinamento em (1) validação e (2) treinamento parcial, tanto X (resenhas) quanto Y (etiquetas).

[10]:
```
valid_x = treino_x[:1000]
treino_x_parcial = treino_x[1000:]
valid_y = treino_y[:1000]
treino_y_parcial = treino_y[1000:]
```

Importação dos módulos e definição da arquitetura da rede.

Desta vez, para tentar representar melhor as 46 classes, vamos aumentar o tamanho das camadas internas.

O vetor de saída terá 46 dimensões, uma para cada classe possível.

A função de ativação na saída é a softmax, que distribui a probabilidade de classficiação entre as 46 classes (isto é, juntas, somam 1).

[11]:
```
from keras import models
from keras import layers

modelo = models.Sequential()
modelo.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
modelo.add(layers.Dense(64, activation='relu'))
modelo.add(layers.Dense(46, activation='softmax'))
```

Compilação do modelo.

[12]:
```
# A diferença dos parâmetros de compilação que usaremos aqui por relação à
# classificação binária está na função de perda escolhida.
# Ainda é a entropia cruzada, boa para classificação probabilística, mas,
# desta vez, é a "categorial", não a binária.

modelo.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['acc'])
```

Treinamento do modelo compilado.

[13]:
```
historia = modelo.fit(treino_x_parcial,
                      treino_y_parcial,
                      epochs=20,
                      batch_size=512,
                      validation_data=(valid_x, valid_y))
```

```
Epoch 1/20
16/16 [==============================] - 2s 73ms/step - loss: 3.5577 - acc:
0.2945 - val_loss: 2.6135 - val_acc: 0.5690
```

```
Epoch 2/20
16/16 [==============================] - 1s 43ms/step - loss: 2.2565 - acc:
0.6092 - val_loss: 1.6517 - val_acc: 0.6590
Epoch 3/20
16/16 [==============================] - 1s 44ms/step - loss: 1.4408 - acc:
0.7011 - val_loss: 1.2696 - val_acc: 0.7290
Epoch 4/20
16/16 [==============================] - 1s 43ms/step - loss: 1.0453 - acc:
0.7704 - val_loss: 1.1228 - val_acc: 0.7690
Epoch 5/20
16/16 [==============================] - 1s 43ms/step - loss: 0.8431 - acc:
0.8172 - val_loss: 1.0340 - val_acc: 0.7860
Epoch 6/20
16/16 [==============================] - 1s 44ms/step - loss: 0.6387 - acc:
0.8668 - val_loss: 0.9618 - val_acc: 0.8090
Epoch 7/20
16/16 [==============================] - 1s 43ms/step - loss: 0.4961 - acc:
0.8978 - val_loss: 0.9301 - val_acc: 0.8100
Epoch 8/20
16/16 [==============================] - 1s 44ms/step - loss: 0.3976 - acc:
0.9191 - val_loss: 0.8987 - val_acc: 0.8140
Epoch 9/20
16/16 [==============================] - 1s 44ms/step - loss: 0.3105 - acc:
0.9358 - val_loss: 0.8953 - val_acc: 0.8180
Epoch 10/20
16/16 [==============================] - 1s 43ms/step - loss: 0.2550 - acc:
0.9467 - val_loss: 0.8936 - val_acc: 0.8240
Epoch 11/20
16/16 [==============================] - 1s 43ms/step - loss: 0.2060 - acc:
0.9504 - val_loss: 0.9131 - val_acc: 0.8150
Epoch 12/20
16/16 [==============================] - 1s 43ms/step - loss: 0.1783 - acc:
0.9589 - val_loss: 0.9185 - val_acc: 0.8180
Epoch 13/20
16/16 [==============================] - 1s 43ms/step - loss: 0.1550 - acc:
0.9584 - val_loss: 0.9307 - val_acc: 0.8200
Epoch 14/20
16/16 [==============================] - 1s 42ms/step - loss: 0.1381 - acc:
0.9592 - val_loss: 0.9547 - val_acc: 0.8050
Epoch 15/20
16/16 [==============================] - 1s 44ms/step - loss: 0.1291 - acc:
0.9578 - val_loss: 0.9597 - val_acc: 0.8160
Epoch 16/20
16/16 [==============================] - 1s 43ms/step - loss: 0.1172 - acc:
0.9611 - val_loss: 0.9661 - val_acc: 0.8110
Epoch 17/20
16/16 [==============================] - 1s 43ms/step - loss: 0.1152 - acc:
0.9603 - val_loss: 0.9960 - val_acc: 0.8130
```

```
Epoch 18/20
16/16 [==============================] - 1s 44ms/step - loss: 0.1049 - acc:
0.9635 - val_loss: 0.9805 - val_acc: 0.8140
Epoch 19/20
16/16 [==============================] - 1s 43ms/step - loss: 0.0949 - acc:
0.9648 - val_loss: 1.0027 - val_acc: 0.8110
Epoch 20/20
16/16 [==============================] - 1s 42ms/step - loss: 0.0992 - acc:
0.9609 - val_loss: 1.0190 - val_acc: 0.8130
```

Exibição da evolução da perda no treinamento e na validação.

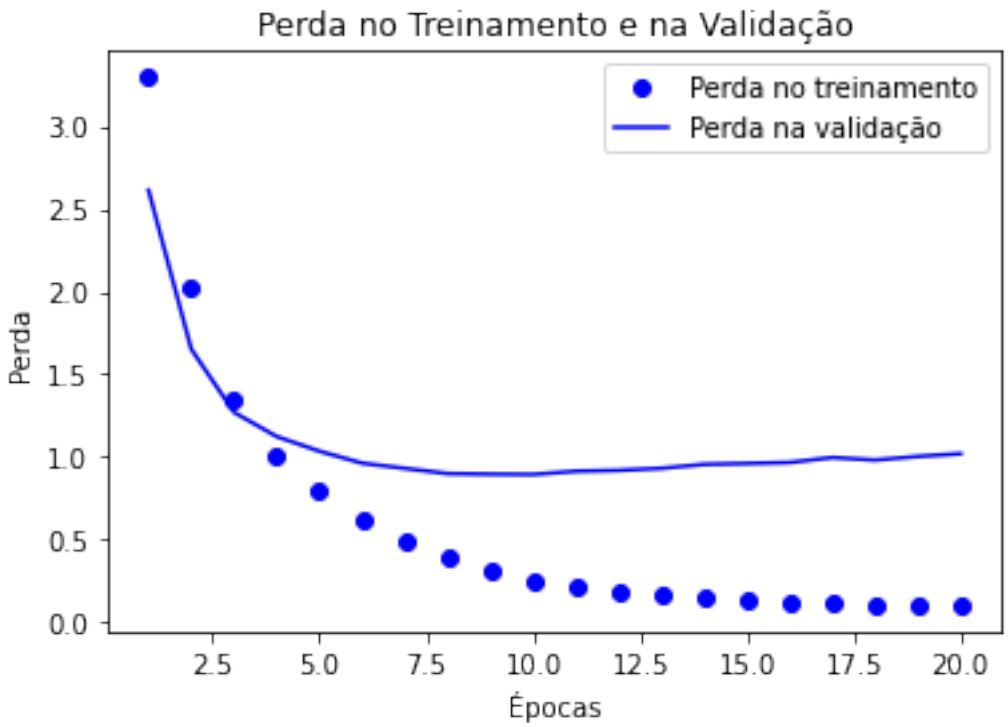
[14]:
```
import matplotlib.pyplot as plt

dic_historia = historia.history  # dict_keys(['loss', 'acc', 'val_loss',␣
 ↪'val_acc'])
perda = dic_historia['loss']
perda_valid = dic_historia['val_loss']

acuracia = dic_historia['acc']
epocas = range(1, len(acuracia) + 1)

plt.plot(epocas, perda, 'bo', label='Perda no treinamento')  # "bo" = pontilhado␣
 ↪azul
plt.plot(epocas, perda_valid, 'b', label='Perda na validação')  # "b" = linha␣
 ↪contínua azul
plt.title('Perda no Treinamento e na Validação')
plt.xlabel('Épocas')
plt.ylabel('Perda')
plt.legend()

plt.show()
```



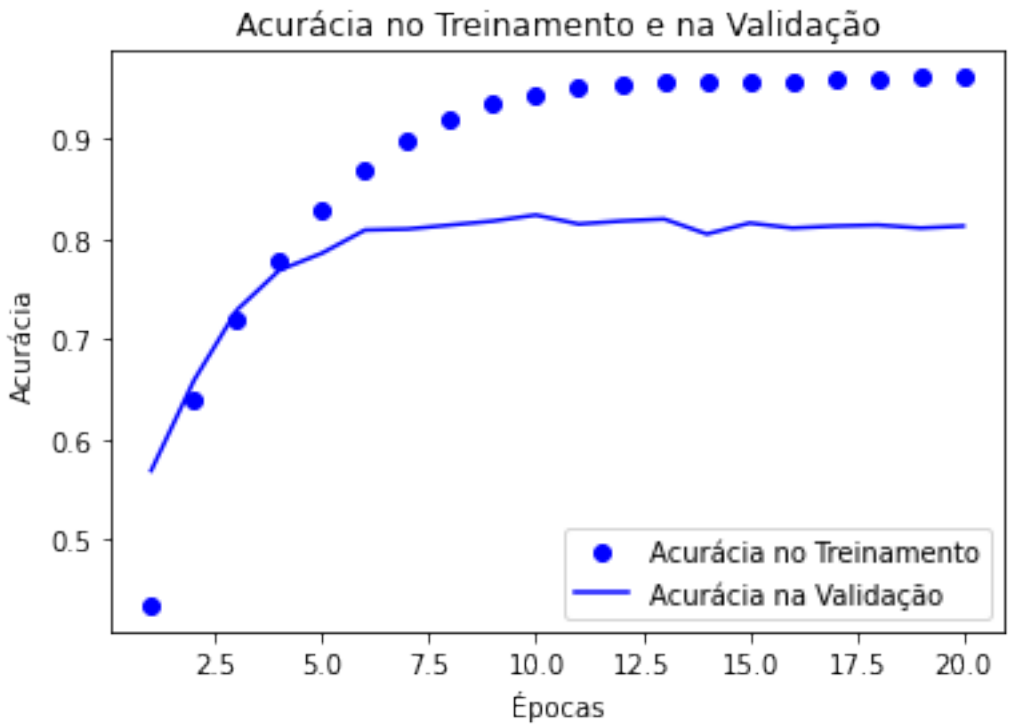
Exibição da evolução da acurácia no teste e na validação.

[15]:
```
plt.clf()  # Limpa a figura

acuracia_treino = dic_historia['acc']
acuracia_valid = dic_historia['val_acc']

plt.plot(epocas, acuracia_treino, 'bo', label='Acurácia no Treinamento')
plt.plot(epocas, acuracia_valid, 'b', label='Acurácia na Validação')
plt.title('Acurácia no Treinamento e na Validação')
plt.xlabel('Épocas')
plt.ylabel('Acurácia')
plt.legend()

plt.show()
```



4 Introdução de um gargalo na rede

Vamos "esmagar" uma camada da rede, reduzindo propositadamente seu tamanho para, em seguida, avaliar os resultados.

[16]:
```
from keras import models
from keras import layers

modelo = models.Sequential()
modelo.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
modelo.add(layers.Dense(4, activation='relu'))  # Camada esmagada!!!
modelo.add(layers.Dense(46, activation='softmax'))
```

[17]:
```
### O resto fica igual...

modelo.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['acc'])

historia = modelo.fit(treino_x_parcial,
                      treino_y_parcial,
                      epochs=20,
                      batch_size=512,
```

```
                    validation_data=(valid_x, valid_y))
```

```
Epoch 1/20
16/16 [==============================] - 1s 54ms/step - loss: 3.6849 - acc:
0.3061 - val_loss: 3.2840 - val_acc: 0.4070
Epoch 2/20
16/16 [==============================] - 1s 44ms/step - loss: 3.1280 - acc:
0.4118 - val_loss: 2.7478 - val_acc: 0.4140
Epoch 3/20
16/16 [==============================] - 1s 43ms/step - loss: 2.5684 - acc:
0.4352 - val_loss: 2.2712 - val_acc: 0.4170
Epoch 4/20
16/16 [==============================] - 1s 43ms/step - loss: 2.1334 - acc:
0.4289 - val_loss: 1.9523 - val_acc: 0.4220
Epoch 5/20
16/16 [==============================] - 1s 42ms/step - loss: 1.7698 - acc:
0.4493 - val_loss: 1.7461 - val_acc: 0.4510
Epoch 6/20
16/16 [==============================] - 1s 43ms/step - loss: 1.5546 - acc:
0.5225 - val_loss: 1.5955 - val_acc: 0.6320
Epoch 7/20
16/16 [==============================] - 1s 43ms/step - loss: 1.3888 - acc:
0.6827 - val_loss: 1.4856 - val_acc: 0.6510
Epoch 8/20
16/16 [==============================] - 1s 43ms/step - loss: 1.2665 - acc:
0.6869 - val_loss: 1.4193 - val_acc: 0.6560
Epoch 9/20
16/16 [==============================] - 1s 43ms/step - loss: 1.1304 - acc:
0.7090 - val_loss: 1.3682 - val_acc: 0.6710
Epoch 10/20
16/16 [==============================] - 1s 43ms/step - loss: 1.0492 - acc:
0.7300 - val_loss: 1.3379 - val_acc: 0.6920
Epoch 11/20
16/16 [==============================] - 1s 43ms/step - loss: 0.9693 - acc:
0.7641 - val_loss: 1.3152 - val_acc: 0.7050
Epoch 12/20
16/16 [==============================] - 1s 43ms/step - loss: 0.9121 - acc:
0.7770 - val_loss: 1.3062 - val_acc: 0.7050
Epoch 13/20
16/16 [==============================] - 1s 42ms/step - loss: 0.8657 - acc:
0.7925 - val_loss: 1.3042 - val_acc: 0.7110
Epoch 14/20
16/16 [==============================] - 1s 44ms/step - loss: 0.8218 - acc:
0.8001 - val_loss: 1.3098 - val_acc: 0.7080
Epoch 15/20
16/16 [==============================] - 1s 43ms/step - loss: 0.7803 - acc:
0.8062 - val_loss: 1.3095 - val_acc: 0.7070
```

```
Epoch 16/20
16/16 [==============================] - 1s 43ms/step - loss: 0.7318 - acc:
0.8131 - val_loss: 1.3134 - val_acc: 0.7070
Epoch 17/20
16/16 [==============================] - 1s 43ms/step - loss: 0.7048 - acc:
0.8189 - val_loss: 1.3338 - val_acc: 0.7090
Epoch 18/20
16/16 [==============================] - 1s 43ms/step - loss: 0.6796 - acc:
0.8184 - val_loss: 1.3402 - val_acc: 0.7080
Epoch 19/20
16/16 [==============================] - 1s 42ms/step - loss: 0.6308 - acc:
0.8288 - val_loss: 1.3586 - val_acc: 0.7070
Epoch 20/20
16/16 [==============================] - 1s 42ms/step - loss: 0.6375 - acc:
0.8210 - val_loss: 1.3721 - val_acc: 0.7100
```

Vamos retreinar o modelo, desafogando o gargalo e inserindo o EarlyStopping por conveniência.

[18]:
```
from tensorflow.keras import callbacks

modelo = models.Sequential()
modelo.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
modelo.add(layers.Dense(64, activation='relu'))
modelo.add(layers.Dense(46, activation='softmax'))

modelo.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['acc'])

aprendeu_parou = callbacks.EarlyStopping(
    min_delta=0.001,  # aprendizado mínimo (resultados menores não contarão como␣
 ↪aprendizado)
    patience=10,  # por quantas épocas insistir?
    restore_best_weights=True)

historia = modelo.fit(treino_x_parcial,
                      treino_y_parcial,
                      epochs=300,
                      batch_size=512,
                      validation_data=(valid_x, valid_y),
                      callbacks=[aprendeu_parou])
```

```
Epoch 1/300
16/16 [==============================] - 1s 53ms/step - loss: 3.4813 - acc:
0.3440 - val_loss: 2.4256 - val_acc: 0.5350
Epoch 2/300
16/16 [==============================] - 1s 44ms/step - loss: 2.1688 - acc:
0.5534 - val_loss: 1.6575 - val_acc: 0.6110
```

```
Epoch 3/300
16/16 [==============================] - 1s 43ms/step - loss: 1.4824 - acc:
0.6740 - val_loss: 1.3715 - val_acc: 0.6810
Epoch 4/300
16/16 [==============================] - 1s 44ms/step - loss: 1.1695 - acc:
0.7366 - val_loss: 1.1996 - val_acc: 0.7330
Epoch 5/300
16/16 [==============================] - 1s 44ms/step - loss: 0.9489 - acc:
0.7917 - val_loss: 1.0901 - val_acc: 0.7600
Epoch 6/300
16/16 [==============================] - 1s 43ms/step - loss: 0.7620 - acc:
0.8275 - val_loss: 1.0180 - val_acc: 0.7780
Epoch 7/300
16/16 [==============================] - 1s 43ms/step - loss: 0.6175 - acc:
0.8698 - val_loss: 0.9653 - val_acc: 0.7930
Epoch 8/300
16/16 [==============================] - 1s 43ms/step - loss: 0.4784 - acc:
0.8999 - val_loss: 0.9227 - val_acc: 0.8000
Epoch 9/300
16/16 [==============================] - 1s 44ms/step - loss: 0.3747 - acc:
0.9214 - val_loss: 0.8917 - val_acc: 0.8160
Epoch 10/300
16/16 [==============================] - 1s 43ms/step - loss: 0.2963 - acc:
0.9378 - val_loss: 0.8972 - val_acc: 0.8080
Epoch 11/300
16/16 [==============================] - 1s 43ms/step - loss: 0.2386 - acc:
0.9457 - val_loss: 0.8938 - val_acc: 0.8090
Epoch 12/300
16/16 [==============================] - 1s 44ms/step - loss: 0.1892 - acc:
0.9553 - val_loss: 0.9090 - val_acc: 0.8120
Epoch 13/300
16/16 [==============================] - 1s 44ms/step - loss: 0.1723 - acc:
0.9562 - val_loss: 0.9167 - val_acc: 0.8130
Epoch 14/300
16/16 [==============================] - 1s 43ms/step - loss: 0.1451 - acc:
0.9599 - val_loss: 0.9371 - val_acc: 0.8100
Epoch 15/300
16/16 [==============================] - 1s 44ms/step - loss: 0.1285 - acc:
0.9616 - val_loss: 0.9465 - val_acc: 0.8040
Epoch 16/300
16/16 [==============================] - 1s 43ms/step - loss: 0.1181 - acc:
0.9629 - val_loss: 0.9573 - val_acc: 0.8120
Epoch 17/300
16/16 [==============================] - 1s 43ms/step - loss: 0.1108 - acc:
0.9651 - val_loss: 0.9812 - val_acc: 0.8060
Epoch 18/300
16/16 [==============================] - 1s 43ms/step - loss: 0.0983 - acc:
0.9670 - val_loss: 0.9952 - val_acc: 0.8070
```

```
Epoch 19/300
16/16 [==============================] - 1s 43ms/step - loss: 0.0998 - acc:
0.9614 - val_loss: 0.9874 - val_acc: 0.8080
```

5 Avaliação

[19]:
```
avaliacao = modelo.evaluate(teste_x, teste_y)
```

```
71/71 [==============================] - 0s 3ms/step - loss: 0.9695 - acc:
0.7863
```

[20]:
```
print('Acurácia na avaliação: ', avaliacao[1], '\nPerda: ', avaliacao[0])
```

```
Acurácia na avaliação:  0.7862867116928101
Perda:  0.9695258140563965
```

6 Previsões (etiquetagens feitas pelo modelo)

[21]:
```
previsoes = modelo.predict(teste_x)
len(previsoes)
```

[21]: 2246

[22]:
```
# Quantas são as classes possíveis para uma notícia qualquer?
previsoes[0].shape
```

[22]: (46,)

[23]:
```
# Soma das probabilidades de todas as classes
sum(previsoes[0])
```

[23]: 0.9999999844012564

[24]:
```
# Qual a classe estimada para uma dada notícia?
# Note que o que está sendo buscado é o índice (i.e., posição no vetor de saída)
# da classe com maior probabilidade

np.argmax(previsoes[0])
```

[24]: 3

[25]:
```
# E qual foi exatamente a probabilidade calculada para essa classe "campeã"?
previsoes[0][3]
```

[25]: 0.86115056

7 Tarefa: clusterização de assuntos

As notícias do conjunto de testes foram classificadas em 46 assuntos.

Agora, você deve implementar:

1. Uma função que agrupe em listas as notícias divididas por assunto (para cada assunto haverá uma lista de notícias).
2. Uma função que receba um número qualquer correspondente à etiqueta de assunto e mostre o texto legível (não os vetores!) de três notícias quaisquer classificadas sob esse assunto.

Aqui está a lista dos 46 assuntos do corpus Reuters:

[26]:
```
assuntos =␣
↪['cocoa','grain','veg-oil','earn','acq','wheat','copper','housing','money-supply',
 ␣
↪'coffee','sugar','trade','reserves','ship','cotton','carcass','crude','nat-gas',
  'cpi','money-fx','interest','gnp','meal-feed','alum','oilseed','gold','tin',
 ␣
↪'strategic-metal','livestock','retail','ipi','iron-steel','rubber','heat','jobs',
 ␣
↪'lei','bop','zinc','orange','pet-chem','dlr','gas','silver','wpi','hog','lead']
```

[27]:
```
len(assuntos)
```

[27]:
```
46
```

Assunto: [noticia1, n2, n3...],

[97]:
```
# exercício 1
from collections import defaultdict

def agrupa_noticias(noticias, predicoes, dicionario_de_assuntos,␣
↪palavra_de_indice):
    """
    A função agrupa notícias recebe as notícias em texto codificado,
    as predições como o output do modelo, a lista correspondente de assuntos
    das predições e o dicionário de índices de cada palavra.
    """
    # conversão das predições como output do modelo para uma lista com a␣
↪descrição do assunto
    predicoes_desc = [dicionario_de_assuntos[np.argmax(predicao)] for predicao␣
↪in predicoes]
    # usando defaultdict conseguimos ter um formato mais limpo de código do que␣
↪se usássemos dicionários nativos
    assuntos_dict = defaultdict(list)
    for i, predicao in enumerate(predicoes_desc):
        # iterando em cada predição o seguinte procedimento é realizado:
```

```
        # dicionário na posição assunto recebe uma lista e coloca ao final␣
 ↪usando o append
        # a lista que será colocada ao final é a notícia em formato descrito,␣
 ↪iterando em cada
        # código de palavra de cada notícia e buscando a descrição␣
 ↪correspondente no dicionário de id_palavras
        assuntos_dict[predicao].append(' '.join([palavra_de_indice.
 ↪get(id_palavra-3, '<?>') for id_palavra in noticias[i]]))
    return assuntos_dict
noticias_agrupadas = agrupa_noticias(dados_teste, previsoes, assuntos,␣
 ↪palavra_de_indice)
# exemplo
print(noticias_agrupadas["jobs"][0])
```

```
<?> the u s civilian unemployment rate fell to 6 6 pct in march from 6 7 pct in
february the labor department said the number of non farm payroll jobs rose 164
000 last month after rising a revised 236 000 in february that was down from the
previously reported 337 000 rise in february the march unemployment rate was the
lowest since march 1980 it had remained unchanged at 6 7 pct for three straight
months before the march decline the rise in non farm <?> was the smallest since
a decline last june of 75 000 the department said last month's unemployment rate
was down from the 7 2 pct level in march 1986 growth in jobs continued in march
but was slower than in recent months with the gains concentrated in service
industries the number of goods producing jobs fell 68 000 in march while service
producing jobs rose 232 000 to bring the total jobs in the department's survey
of businesses to 102 03 mln in march business and health services showed the
largest gains in jobs while manufacturing employment fell by 25 000 the average
work week fell to 34 8 hours in march from 35 0 hours in february the department
said manufacturing hours fell to 40 9 per week from 41 2 hours in february but
overtime hours increased to 3 7 from 3 6 the department's survey of households
showed the number of unemployed stood at 7 85 mln out of a work force of 119 2
mln the number of persons working part time for economic reasons fell in march
to 5 46 mln from 5 78 mln in february the loss of factory jobs brought the march
total to 19 19 mln jobs and was concentrated in automobile electrical and
electronic manufacturing construction employment also lowered the number of jobs
in the goods producing sector falling by 45 000 after seasonal adjustment the
department said mining employment was little changed in march and has not
experienced any substantial erosion since the rapid job losses in oil and gas
drilling in the first two thirds of 1986 other service industries that increased
jobs last month were finance insurance and real estate reuter 3
```

[103]:
```
# exercício 2

def mostra_k_noticias(noticias_agrupadas, grupo, k=3):
    """
    Essa função recebe o dicionário criado na função
    anterior e mostra as k primeiras notícias
```

```
    do assunto do grupo selecionado
    """

    if k > len(noticias_agrupadas[grupo]):
        raise ValueError(f"Você pediu por {k} notícias, mas esse assunto só␣
↪possui {len(noticias_agrupadas[grupo])} notícias")

    print(f"Assunto escolhido: {grupo}\n-----------------------")
    for i in range(k):
        noticia_junta = noticias_agrupadas[grupo][i]
        print(noticia_junta)

mostra_k_noticias(noticias_agrupadas, "jobs", 3)
```

```
Assunto escolhido: jobs
-----------------------
<?> the u s civilian unemployment rate fell to 6 6 pct in march from 6 7 pct in
february the labor department said the number of non farm payroll jobs rose 164
000 last month after rising a revised 236 000 in february that was down from the
previously reported 337 000 rise in february the march unemployment rate was the
lowest since march 1980 it had remained unchanged at 6 7 pct for three straight
months before the march decline the rise in non farm <?> was the smallest since
a decline last june of 75 000 the department said last month's unemployment rate
was down from the 7 2 pct level in march 1986 growth in jobs continued in march
but was slower than in recent months with the gains concentrated in service
industries the number of goods producing jobs fell 68 000 in march while service
producing jobs rose 232 000 to bring the total jobs in the department's survey
of businesses to 102 03 mln in march business and health services showed the
largest gains in jobs while manufacturing employment fell by 25 000 the average
work week fell to 34 8 hours in march from 35 0 hours in february the department
said manufacturing hours fell to 40 9 per week from 41 2 hours in february but
overtime hours increased to 3 7 from 3 6 the department's survey of households
showed the number of unemployed stood at 7 85 mln out of a work force of 119 2
mln the number of persons working part time for economic reasons fell in march
to 5 46 mln from 5 78 mln in february the loss of factory jobs brought the march
total to 19 19 mln jobs and was concentrated in automobile electrical and
electronic manufacturing construction employment also lowered the number of jobs
in the goods producing sector falling by 45 000 after seasonal adjustment the
department said mining employment was little changed in march and has not
experienced any substantial erosion since the rapid job losses in oil and gas
drilling in the first two thirds of 1986 other service industries that increased
jobs last month were finance insurance and real estate reuter 3

<?> spain's registered unemployment fell by 10 465 people to 2 97 mln or 21 4
pct of the workforce in march labour ministry figures show registered
unemployment in february was 2 98 mln people or 21 5 pct of the workforce the
figures were nonetheless higher than those for march 1986 2 8 mln people and 21
```

pct of the workforce reuter 3

<?> japan's seasonally adjusted unemployment rate fell to 2 9 pct in february from the record 3 0 pct in january the government's management and coordination agency said the january level was the worst since the goverment started compiling unemployment statistics under the current system in 1953 unemployment was up from 2 8 pct a year earlier unadjusted february unemployment totalled 1 86 mln people up from 1 82 mln in january and 1 64 mln a year earlier male unemployment in february remained at 2 9 pct equal to the second highest level set in january and december record male unemployment of 3 1 pct was set in july 1986 female unemployment rose to a record 3 1 pct in february from the previous record 3 0 pct marked in january 1987 and in april august september and december last year japan's employment condition was still severe in february as the non <?> rate of unemployment in february fell only 0 03 percentage points to 2 93 pct from 2 96 pct in january an agency official said employment in manufacturing industries fell 430 000 from a year earlier to 14 22 mln people in february due to the yen's continued appreciation while employment in non manufacturing industries rose 380 000 to 12 11 mln in manufacturing industries employment in the textile industry fell 180 000 to 1 94 mln in february while in ordinary and precision machinery industries it fell 160 000 to 1 50 mln reuter 3