# Classifying newswires: a multiclass classification example

We will be working with the Reuters dataset, a set of short newswires and their topics, published by Reuters in 1986. It's a very simple, widely used toy dataset for text classification. There are 46 different topics; some topics are more represented than others, but each topic has at least 10 examples in the training set.

**from keras.datasets import reuters**

**(train\_data, train\_labels), (test\_data, test\_labels) = reuters.load\_data(num\_words = 10000)**

**Downloading data from https://s3.amazonaws.com/text-datasets/reuters.npz**

**2113536/2110848 [==============================] - 1s 0us/step**

Like with the IMDB dataset, the argument num\_words=10000 restricts the data to the 10,000 most frequently occurring words found in the data.

We have 8,982 training examples and 2,246 test examples:

**len(train\_data)**

**8982**

Here's how you can decode it back to words, in case you are curious:

**word\_index = reuters.get\_word\_index()**

**reverse\_word\_index = dict([(value, key) for (key, value) in word\_index.items()])**

**# Note that our indices were offset by 3**

**# because 0, 1 and 2 are reserved indices for "padding", "start of sequence", and "unknown".**

**decoded\_newswire = ' '.join([reverse\_word\_index.get(i - 3, '?') for i in train\_data[0]])**

**Downloading data from https://s3.amazonaws.com/text-datasets/reuters\_word\_index.json**

**557056/550378 [==============================] - 0s 0us/step**

**decoded\_newswire**

'? ? ? 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'

## **Preparing the data**

We can vectorize the data with the exact same code as in our previous example:

**import numpy as np**

**def vectorize\_sequences(sequences, dimension=10000):**

**results = np.zeros((len(sequences), dimension))**

**for i, sequence in enumerate(sequences):**

**results[i, sequence] = 1.**

**return results**

**# Our vectorized training data**

**x\_train = vectorize\_sequences(train\_data)**

**# Our vectorized test data**

**x\_test = vectorize\_sequences(test\_data)**

**def to\_one\_hot(labels, dimension=46):**

**results = np.zeros((len(labels), dimension))**

**for i, label in enumerate(labels):**

**results[i, label] = 1.**

**return results**

**# Our vectorized training labels**

**one\_hot\_train\_labels = to\_one\_hot(train\_labels)**

**# Our vectorized test labels**

**one\_hot\_test\_labels = to\_one\_hot(test\_labels)**

**from keras.utils.np\_utils import to\_categorical**

**one\_hot\_train\_labels = to\_categorical(train\_labels)**

**one\_hot\_test\_labels = to\_categorical(test\_labels)**

## **Building our network**

This topic classification problem looks very similar to our previous movie review classification problem: in both cases, we are trying to classify short snippets of text. There is however a new constraint here: the number of output classes has gone from 2 to 46, i.e. the dimensionality of the output space is much larger.

**from keras import models**

**from keras import layers**

**model = models.Sequential()**

**model.add(layers.Dense(64, activation='relu', input\_shape=(10000,)))**

**model.add(layers.Dense(64, activation='relu'))**

**model.add(layers.Dense(46, activation='softmax'))**

The best loss function to use in this case is categorical\_crossentropy. It measures the distance between two probability distributions: in our case, between the probability distribution output by our network, and the true distribution of the labels. By minimizing the distance between these two distributions, we train our network to output something as close as possible to the true labels.

**model.compile(optimizer='rmsprop',**

**loss='categorical\_crossentropy',**

**metrics=['accuracy'])**

## **Validating our approach**

Let's set apart 1,000 samples in our training data to use as a validation set and train our network for 20 Epochs:

**x\_val = x\_train[:1000]**

**partial\_x\_train = x\_train[1000:]**

**y\_val = one\_hot\_train\_labels[:1000]**

**partial\_y\_train = one\_hot\_train\_labels[1000:]**

**history = model.fit(partial\_x\_train,**

**partial\_y\_train,**

**epochs=20,**

**batch\_size=512,**

**validation\_data=(x\_val, y\_val))**

**Train on 7982 samples, validate on 1000 samples**

**Epoch 1/20**

**7982/7982 [==============================] - 2s 227us/step - loss: 2.5322 - acc:**

**Epoch 20/20**

**7982/7982 [==============================] - 1s 147us/step - loss: 0.1111 - acc: 0.9594 - val\_loss: 1.0728 - val\_acc: 0.8010**

**import matplotlib.pyplot as plt**

**loss = history.history['loss']**

**val\_loss = history.history['val\_loss']**

**epochs = range(1, len(loss) + 1)**

**plt.plot(epochs, loss, 'bo', label='Training loss')**

**plt.plot(epochs, val\_loss, 'b', label='Validation loss')**

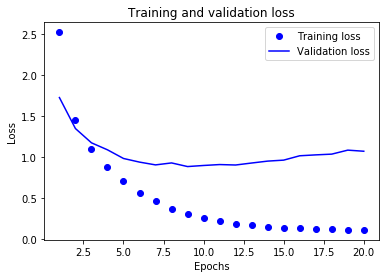
**plt.title('Training and validation loss')**

**plt.xlabel('Epochs')**

**plt.ylabel('Loss')**

**plt.legend()**

**plt.show()**

****

**plt.clf() # clear figure**

**acc = history.history['acc']**

**val\_acc = history.history['val\_acc']**

**plt.plot(epochs, acc, 'bo', label='Training acc')**

**plt.plot(epochs, val\_acc, 'b', label='Validation acc')**

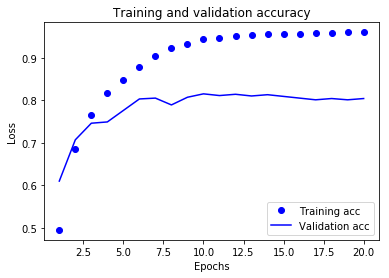
**plt.title('Training and validation accuracy')**

**plt.xlabel('Epochs')**

**plt.ylabel('Loss')**

**plt.legend()**

**plt.show()**

****

It seems that the network starts overfitting after 8 epochs. Let's train a new network from scratch for 8 epochs, then let's evaluate it on the test set:

**model = models.Sequential()**

**model.add(layers.Dense(64, activation='relu', input\_shape=(10000,)))**

**model.add(layers.Dense(64, activation='relu'))**

**model.add(layers.Dense(46, activation='softmax'))**

**model.compile(optimizer='rmsprop',**

**loss='categorical\_crossentropy',**

**metrics=['accuracy'])**

**model.fit(partial\_x\_train,**

**partial\_y\_train,**

**epochs=8,**

**batch\_size=512,**

**validation\_data=(x\_val, y\_val))**

**results = model.evaluate(x\_test, one\_hot\_test\_labels)**

**Train on 7982 samples, validate on 1000 samples**

**Epoch 1/8**

**7982/7982 [==============================] - 1s 154us/step - loss: 2.5427 - acc: 0.5231 - val\_loss: 1.6855 - val\_acc: 0.6500**

**Epoch 8/8**

**7982/7982 [==============================] - 1s 141us/step - loss: 0.3374 - acc: 0.9278 - val\_loss: 0.8784 - val\_acc: 0.8240**

**2246/2246 [==============================] - 0s 92us/step**

**results**

**[0.9870049482143467, 0.7818343722172751]**