Audiobooks business case

Extract the data from the csv

In [54]:
```
import numpy as np
from sklearn import preprocessing
import numpy as np
import tensorflow as tf

raw_csv_data = np.loadtxt('Audiobooks_data.csv',delimiter=',')
unscaled_inputs_all = raw_csv_data[:,0:-1]
targets_all = raw_csv_data[:,-1]
```

Balance the dataset

In [55]:
```
num_one_targets = int(np.sum(targets_all))

zero_targets_counter = 0

indices_to_remove = []

for i in range(targets_all.shape[0]):
    if targets_all[i] == 0:
        zero_targets_counter += 1
        if zero_targets_counter > num_one_targets:
            indices_to_remove.append(i)

# Create two new variables, one that will contain the inputs, and one that will c
# We delete all indices that we marked "to remove" in the loop above.
unscaled_inputs_equal_priors = np.delete(unscaled_inputs_all, indices_to_remove,
targets_equal_priors = np.delete(targets_all, indices_to_remove, axis=0)
```

Standardize the inputs

In [56]:
```
scaled_inputs = preprocessing.scale(unscaled_inputs_equal_priors)
# print(scaled_inputs)
```

Shuffle the data

In [57]:
```
# Shuffle the indices of the data, so the data is not arranged in any way when we
shuffled_indices = np.arange(scaled_inputs.shape[0])
np.random.shuffle(shuffled_indices)

# Use the shuffled indices to shuffle the inputs and targets.
shuffled_inputs = scaled_inputs[shuffled_indices]
shuffled_targets = targets_equal_priors[shuffled_indices]
```

Split the dataset into train, validation, and test

In [58]:
```
samples_count = shuffled_inputs.shape[0]

train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1 * samples_count)

test_samples_count = samples_count - train_samples_count - validation_samples_cou

train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]

validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+valic
validation_targets = shuffled_targets[train_samples_count:train_samples_count+val

test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]


# Print the number of targets that are 1s, the total number of samples, and the p
print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_s
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_tar
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_sampl
```

```
1806.0 3579 0.5046102263202011
223.0 447 0.4988814317673378
208.0 448 0.4642857142857143
```

Save/Load the three datasets in *.npz (Optional)

In [59]:
```
# Save the three datasets in *.npz.
# In the next lesson, you will see that it is extremely valuable to name them in

np.savez('Audiobooks_data_train', inputs=train_inputs, targets=train_targets)
np.savez('Audiobooks_data_validation', inputs=validation_inputs, targets=validati
np.savez('Audiobooks_data_test', inputs=test_inputs, targets=test_targets)
```

In [60]:
```
npz = np.load('Audiobooks_data_train.npz')

train_inputs = npz['inputs'].astype(float)
train_targets = npz['targets'].astype(int)

npz = np.load('Audiobooks_data_validation.npz')
# we can load the inputs and the targets in the same line
validation_inputs, validation_targets = npz['inputs'].astype(float), npz['targets

# we load the test data in the temporary variable
npz = np.load('Audiobooks_data_test.npz')
# we create 2 variables that will contain the test inputs and the test targets
test_inputs, test_targets = npz['inputs'].astype(float), npz['targets'].astype(in
```

Model Outline

In [61]:
```
# Set the input and output sizes
input_size = 10
output_size = 2
# Use same hidden layer size for both hidden layers. Not a necessity.
hidden_layer_size = 25

# define how the model will look like
model = tf.keras.Sequential([
    # tf.keras.layers.Dense is basically implementing: output = activation(dot(ir
    # it takes several arguments, but the most important ones for us are the hidd
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'), # 1st hidden lay
    tf.keras.layers.Dense(hidden_layer_size, activation='tanh'), # 1st hidden lay
    # the final layer is no different, we just make sure to activate it with soft
    tf.keras.layers.Dense(output_size, activation='sigmoid') # output layer
])


### Choose the optimizer and the loss function

# we define the optimizer we'd like to use,
# the loss function,
# and the metrics we are interested in obtaining at each iteration
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=[
# model.compile(optimizer='adam', loss='MeanAbsoluteError', metrics=['accuracy'])


### Training
# That's where we train the model we have built.

# set the batch size
batch_size = 100

# set a maximum number of training epochs
max_epochs = 1000

# set an early stopping mechanism
# let's set patience=2, to be a bit tolerant against random validation loss incre
early_stopping = tf.keras.callbacks.EarlyStopping(patience=2)

# fit the model
# note that this time the train, validation and test data are not iterable
model.fit(train_inputs, # train inputs
          train_targets, # train targets
          batch_size=batch_size, # batch size
          epochs=max_epochs, # epochs that we will train for (assuming early stop
          # callbacks are functions called by a task when a task is completed
          # task here is to check if val_loss is increasing
          callbacks=[early_stopping], # early stopping
          validation_data=(validation_inputs, validation_targets), # validation d
          verbose = 2 # making sure we get enough information about the training
          )
```

```
Epoch 1/1000
36/36 - 1s - loss: 0.5857 - accuracy: 0.7228 - val_loss: 0.4785 - val_accuracy:
0.8747 - 697ms/epoch - 19ms/step
Epoch 2/1000
36/36 - 0s - loss: 0.4125 - accuracy: 0.8740 - val_loss: 0.3546 - val_accuracy:
0.8949 - 66ms/epoch - 2ms/step
Epoch 3/1000
36/36 - 0s - loss: 0.3199 - accuracy: 0.8877 - val_loss: 0.2980 - val_accuracy:
0.8993 - 62ms/epoch - 2ms/step
Epoch 4/1000
36/36 - 0s - loss: 0.2798 - accuracy: 0.8991 - val_loss: 0.2748 - val_accuracy:
0.9060 - 61ms/epoch - 2ms/step
Epoch 5/1000
36/36 - 0s - loss: 0.2608 - accuracy: 0.9061 - val_loss: 0.2583 - val_accuracy:
0.9060 - 62ms/epoch - 2ms/step
Epoch 6/1000
36/36 - 0s - loss: 0.2466 - accuracy: 0.9072 - val_loss: 0.2473 - val_accuracy:
0.9128 - 67ms/epoch - 2ms/step
Epoch 7/1000
36/36 - 0s - loss: 0.2363 - accuracy: 0.9137 - val_loss: 0.2381 - val_accuracy:
0.9239 - 62ms/epoch - 2ms/step
Epoch 8/1000
36/36 - 0s - loss: 0.2280 - accuracy: 0.9179 - val_loss: 0.2278 - val_accuracy:
0.9195 - 66ms/epoch - 2ms/step
Epoch 9/1000
36/36 - 0s - loss: 0.2182 - accuracy: 0.9204 - val_loss: 0.2181 - val_accuracy:
0.9262 - 63ms/epoch - 2ms/step
Epoch 10/1000
36/36 - 0s - loss: 0.2102 - accuracy: 0.9204 - val_loss: 0.2110 - val_accuracy:
0.9217 - 60ms/epoch - 2ms/step
Epoch 11/1000
36/36 - 0s - loss: 0.2036 - accuracy: 0.9271 - val_loss: 0.2024 - val_accuracy:
0.9239 - 62ms/epoch - 2ms/step
Epoch 12/1000
36/36 - 0s - loss: 0.1980 - accuracy: 0.9276 - val_loss: 0.1958 - val_accuracy:
0.9172 - 59ms/epoch - 2ms/step
Epoch 13/1000
36/36 - 0s - loss: 0.1922 - accuracy: 0.9285 - val_loss: 0.1889 - val_accuracy:
0.9306 - 70ms/epoch - 2ms/step
Epoch 14/1000
36/36 - 0s - loss: 0.1873 - accuracy: 0.9310 - val_loss: 0.1848 - val_accuracy:
0.9329 - 62ms/epoch - 2ms/step
Epoch 15/1000
36/36 - 0s - loss: 0.1845 - accuracy: 0.9324 - val_loss: 0.1809 - val_accuracy:
0.9306 - 61ms/epoch - 2ms/step
Epoch 16/1000
36/36 - 0s - loss: 0.1803 - accuracy: 0.9315 - val_loss: 0.1791 - val_accuracy:
0.9306 - 61ms/epoch - 2ms/step
Epoch 17/1000
36/36 - 0s - loss: 0.1776 - accuracy: 0.9315 - val_loss: 0.1731 - val_accuracy:
0.9396 - 63ms/epoch - 2ms/step
Epoch 18/1000
36/36 - 0s - loss: 0.1754 - accuracy: 0.9327 - val_loss: 0.1728 - val_accuracy:
0.9396 - 60ms/epoch - 2ms/step
Epoch 19/1000
36/36 - 0s - loss: 0.1730 - accuracy: 0.9343 - val_loss: 0.1714 - val_accuracy:
0.9374 - 63ms/epoch - 2ms/step
```

```
Epoch 20/1000
36/36 - 0s - loss: 0.1705 - accuracy: 0.9371 - val_loss: 0.1688 - val_accuracy:
0.9396 - 66ms/epoch - 2ms/step
Epoch 21/1000
36/36 - 0s - loss: 0.1686 - accuracy: 0.9377 - val_loss: 0.1679 - val_accuracy:
0.9396 - 73ms/epoch - 2ms/step
Epoch 22/1000
36/36 - 0s - loss: 0.1670 - accuracy: 0.9352 - val_loss: 0.1702 - val_accuracy:
0.9374 - 73ms/epoch - 2ms/step
Epoch 23/1000
36/36 - 0s - loss: 0.1646 - accuracy: 0.9355 - val_loss: 0.1651 - val_accuracy:
0.9418 - 60ms/epoch - 2ms/step
Epoch 24/1000
36/36 - 0s - loss: 0.1630 - accuracy: 0.9357 - val_loss: 0.1639 - val_accuracy:
0.9463 - 62ms/epoch - 2ms/step
Epoch 25/1000
36/36 - 0s - loss: 0.1624 - accuracy: 0.9394 - val_loss: 0.1673 - val_accuracy:
0.9374 - 62ms/epoch - 2ms/step
Epoch 26/1000
36/36 - 0s - loss: 0.1607 - accuracy: 0.9385 - val_loss: 0.1633 - val_accuracy:
0.9418 - 61ms/epoch - 2ms/step
Epoch 27/1000
36/36 - 0s - loss: 0.1595 - accuracy: 0.9419 - val_loss: 0.1658 - val_accuracy:
0.9396 - 62ms/epoch - 2ms/step
Epoch 28/1000
36/36 - 0s - loss: 0.1581 - accuracy: 0.9427 - val_loss: 0.1634 - val_accuracy:
0.9396 - 60ms/epoch - 2ms/step
```

Out[61]:
```
<keras.src.callbacks.History at 0x2d74df8e170>
```

Test the model

In [62]:
```python
test_loss, test_accuracy = model.evaluate(test_inputs, test_targets)
print('\nTest loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss, test_accu
```

```
14/14 [==============================] - 0s 1ms/step - loss: 0.1668 - accuracy:
0.9509

Test loss: 0.17. Test accuracy: 95.09%
```

In [ ]:
```
```

In [ ]:
```
```