# Practice 5: Sentiment Analysis

The dataset was created for the paper 'From Group to Individual Labels using Deep Features', Kotzias et. al,. KDD 2015. It contains sentences labeled with positive or negative sentiment, extracted from reviews of products, movies, and restaurants. The sentences come from three different websites/fields: imdb.com, amazon.com and yelp.com.

For each website, there exists 500 positive and 500 negative sentences. Those were selected randomly for larger datasets of reviews. Score is either 1 (for positive) or 0 (for negative). The data is compiled in the file `sa.csv`.

```
sa.dat <- read.csv2("sa.csv", header = F)
colnames(sa.dat) <- c("Text", "Sentiment")
sa.dat[,2] <- factor(sa.dat[,2])
dim(sa.dat)
```

```
## [1] 2000    2
```

```
str(sa.dat)
```

```
## 'data.frame':    2000 obs. of  2 variables:
##  $ Text     : chr  "Wow... Loved this place." "Crust is not good." "Not tasty and the texture was jus
##  $ Sentiment: Factor w/ 2 levels "0","1": 2 1 1 2 2 1 1 1 2 2 ...
```

```
head(sa.dat)
```

```
##                                                                                 Text
## 1                                                            Wow... Loved this place.
## 2                                                                  Crust is not good.
## 3                                                 Not tasty and the texture was just nasty.
## 4 Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.
## 5                            The selection on the menu was great and so were the prices.
## 6                                        Now I am getting angry and I want my damn pho.
##   Sentiment
## 1         1
## 2         0
## 3         0
## 4         1
## 5         1
## 6         0
```

The sentences are tokenized with max sentence length 25 and max number of words to include 5000.

```
max_features <- 5000  # Max number of words to include
maxlen <- 25 # sequences longer than this will be filtered out
```

```
x <- pad_sequences(sa.dat[,1],maxlen=maxlen)  # 2D integer tensor of shape (samples, maxlen)
dim(x)
```

The data is separated into training and testing data, following a 75%/25% ratio.

```
x_train[1,]  # the last twenty (maxlen)
```

The network will learn 8-dimensional embeddings for each of the 10,000 words, turn the input integer

sequences (2D integer tensor) into embedded sequences (3D float tensor), flatten the tensor to 2D, and train a single dense layer on top for classification.

```
model<-keras_model_sequential() %>%
  layer_embedding(input_dim = 10000, output_dim = 8, input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 1, activation = "sigmoid")


summary(model)
```

Notice that the embedding layer have 80000=10000x8 weights. Thus, network learns 8-dimensional embeddings for each of the 10000 words.

```
model %>% compile(
  optimizer =  "rmsprop",
  loss = "binary_crossentropy",
  metric = c("acc")
)
```

```
history <- model %>% fit(
  x_train,y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
```

```
plot(history)
```

You get to a validation accuracy of ~76%, which is pretty good considering that you're only looking at 20 words from each review. But note that merely flattening the embedded sequences and training a single dense layer on top leads to a model that treats each word in the input sequence separately, without considering inter-word relationships and sentence structure (for example, this model would likely treat both "this movie is a bomb" and "this movie is the bomb" as being negative reviews). It's much better to add recurrent layers or 1D convolutional layers on top of the embedded sequences to learn features that take into account each sequence as a whole. That's what we'll focus on in the next few sections.

## Pretrained word embeddings

Sometimes, you have so little training data available that you can't use your data alone to learn an appropriate task-specific embedding of your vocabulary. What do you do then? Instead of learning word embeddings jointly with the problem you want to solve, you can load embedding vectors from a precomputed embedding space that you know is highly structured and exhibits useful properties—that captures generic aspects of language structure. The rationale behind using pretrained word embeddings in natural language processing is much the same as for using pretrained convnets in image classification: you don't have enough data available to learn truly powerful features on your own, but you expect the features that you need to be fairly generic—that is, common visual features or semantic features. In this case, it makes sense to reuse features learned on a different problem.

Such word embeddings are generally computed using word-occurrence statistics (observations about what words co-occur in sentences or documents), using a variety of techniques, some involving neural networks, others not. The idea of a dense, low-dimensional embedding space for words, computed in an unsupervised way, was initially explored by Bengio et al. in the early 2000s, but it only started to take off in research and industry applications after the release of one of the most famous and successful word-embedding schemes: the Word2vec algorithm, developed by Tomas Mikolov at Google in 2013.

There are various precomputed databases of word embeddings that you can download and use in a Keras

embedding layer. Word2vec is one of them. Another popular one is called Global Vectors for Word Representation (GloVe), which was developed by Stanford researchers in 2014. This embedding technique is based on factorizing a matrix of word co-occurrence statistics. Its developers have made available precomputed embeddings for millions of English tokens, obtained from Wikipedia data and Common Crawl data.

Let's look at how you can get started using GloVe embeddings in a Keras model. The same method will of course be valid for Word2vec embeddings or any other word-embedding database. You'll also use this example to refresh the text-tokenization techniques we introduced a few paragraphs ago: you'll start from raw text and work your way up.

First, head to ai.stanford.edu/~amaas/data/sentiment and download the raw IMDB dataset (if the URL isn't working anymore, Google "IMDB dataset"). Uncompress it. Now, let's collect the individual training reviews into a list of strings, one string per review. You'll also collect the review labels (positive / negative) into a labels list.

**From raw text to word embeddings**

```
imdb_dir<-"~/Docencia/curs21_22/UB/MESIO/DL/unitat5_RNN/imdb/aclImdb"
train_dir<-file.path(imdb_dir,"train")

labels<-c()
texts<-c()

# Takes a couple of minutes
for (label_type in c("neg", "pos")) {
  label <- switch(label_type, neg = 0, pos = 1)
  dir_name <- file.path(train_dir, label_type)
  for (fname in list.files(dir_name,full.names = TRUE)) {
    texts <- c(texts, readChar(fname, file.info(fname)$size))
    labels <- c(labels, label)
  }
}
```

```
texts[1]
labels[1]
```

```
maxlen <- 50
max_words <- 10000
```

*Tokenizing the data*

```
tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)
sequences <- texts_to_sequences(tokenizer, texts)
word_index = tokenizer$word_index  # les paraules que tenim en la nostra base de dades
#names(word_index)

sequences[1]

cat("Found", length(word_index), "unique tokens.\n")


data <- pad_sequences(sequences, maxlen = maxlen)
```

```
dim(data)
data[1,]
```

```
labels <- as.array(labels)
cat("Shape of data tensor:", dim(data), "\n")
cat('Shape of label tensor:', dim(labels), "\n")
```

```
training_samples <- 200     # small subset of examples
validation_samples <- 10000

indices <- sample(1:nrow(data))
training_indices <- indices[1:training_samples]
validation_indices <- indices[(training_samples + 1):
                              (training_samples + validation_samples)]
x_train <- data[training_indices,]
y_train <- labels[training_indices]
x_val <- data[validation_indices,]
y_val <- labels[validation_indices]
```

*Word embedding*

Go to nlp.stanford.edu/projects/glove, and download the precomputed embeddings from 2014 English Wikipedia. It's a 822 MB zip file called glove.6B.zip, containing 100-dimensional embedding vectors for 400,000 words (or nonword tokens). Unzip it.

```
glove_dir<-"~/Docencia/curs21_22/UB/MESIO/DL/unitat5_RNN/imdb/glove6B"
lines <- readLines(file.path(glove_dir, "glove.6B.100d.txt"))

# el 100-vector de coordenades de 400000 paraules en angles
#lines[50] # les coordenades del after


embeddings_index <- new.env(hash = TRUE, parent = emptyenv())

# processat per obtenir un format llista del word embedding
for (i in 1:length(lines)) {
  line <- lines[[i]]
  values <- strsplit(line, " ")[[1]]
  word <- values[[1]]  # la paraula
  embeddings_index[[word]] <- as.double(values[-1]) # les coordenades de la paraula
}
cat("Found", length(embeddings_index), "word vectors.\n")
```

Examples

```
embeddings_index[["after"]]
embeddings_index[["is"]]
embeddings_index[["sky"]]
```

Next, you'll build an embedding matrix that you can load into an embedding layer. It must be a matrix of shape (max_words, embedding_dim), where each entry $i$ contains the embedding_dim-dimensional vector for the word of index $i$ in the reference word index (built during tokenization). Note that index 1 isn't supposed to stand for any word or token—it's a placeholder.

```
embedding_dim <- 100  # dimension compatible with pretrained embedding
embedding_matrix <- array(0, c(max_words, embedding_dim))
for (word in names(word_index)) {
  index <- word_index[[word]]
  if (index < max_words) {
```

```
    embedding_vector <- embeddings_index[[word]]
    if (!is.null(embedding_vector))
      embedding_matrix[index+1,] <- embedding_vector
  }
}
```

```
dim(embedding_matrix)
```

*Defining the model*

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
summary(model)
```

Param of layer embedding are 1000000 equal to max_words=10000 times embedding_dim=100.

The embedding layer has a single weight matrix: a 2D float matrix where each entry i is the word vector meant to be associated with index $i$. Simple enough. Load the GloVe matrix you prepared into the embedding layer, the first layer in the model.

Additionally, you'll freeze the weights of the embedding layer, following the same rationale you're already familiar with in the context of pretrained convnet features: when parts of a model are pretrained (like your embedding layer) and parts are randomly initialized (like your classifier), the pretrained parts shouldn't be updated during training, to avoid forgetting what they already know. The large gradient updates triggered by the randomly initialized layers would be disruptive to the already-learned features.

```
get_layer(model, index = 1) %>%
  set_weights(list(embedding_matrix)) %>%
  freeze_weights()
```

```
summary(model)
```

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
```

```
history <- model %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
```

```
plot(history)
```

```
save_model_weights_hdf5(model, "pre_trained_glove_model.h5")
```

*Evaluating the model on the test set*

```
test_dir <- file.path(imdb_dir, "test")
labels <- c()
texts <- c()
```

```
for (label_type in c("neg", "pos")) {
  label <- switch(label_type, neg = 0, pos = 1)
  dir_name <- file.path(test_dir, label_type)
  for (fname in list.files(dir_name, pattern = glob2rx("*.txt"),
                           full.names = TRUE)) {
    texts <- c(texts, readChar(fname, file.info(fname)$size))
    labels <- c(labels, label)
  }
}


sequences <- texts_to_sequences(tokenizer, texts)
x_test <- pad_sequences(sequences, maxlen = maxlen)
y_test <- as.array(labels)


model %>%
  load_model_weights_hdf5("pre_trained_glove_model.h5") %>%
  evaluate(x_test, y_test)
```