Basilica R Client Documentation

Release 0.0.2

Jorge Silva

May 31, 2019

Contents

CHAPTER 1

Quickstart

1.1 Install the R client

First, install the R client.

```
install.packages("https://storage.googleapis.com/basilica-r-client/basilica_0.0.1.tar.gz", repos=NULL)
```

1.2 Embed some sentences

Let's embed some sentences to make sure the client is working.

```
library('basilica')
conn <- connect("SLOW_DEMO_KEY")

sentences = c(
    "This is a sentence!",
    "This is a similar sentence!",
    "I don't think this sentence is very similar at all..."
)
embeddings <- embed_sentences(sentences, conn=conn)
print(embeddings)
```

```
[[0.8556405305862427, ...], ...]
```

Let's also make sure these embeddings make sense, by checking that the cosine distance between the two similar sentences is smaller:

```
print(cor(embeddings[1,], embeddings[2,]))
print(cor(embeddings[1,], embeddings[3,]))
```

```
0.8048559
0.6877435
```

Great!

1.3 Get an API key

The example above uses the slow demo key. You can get an API key of your own by signing up at https://www.basilica.ai/accounts/register . (If you already have an account, you can view your API keys at https://www.basilica.ai/api-keys .)

1.4 What next?

- Read the documentation for the client: *Basilica R Client*
- See an in-depth tutorial on training an image model: How To Train An Image Model With Basilica

CHAPTER 2

Basilica R Client

basilica.**connect**(*auth_key*, *server*)

Instantiates and returns a Basilica connection tied to a specific auth key and server. It also populates a global *basilica_connection* that is a copy of the returned connection. If a *conn* argument is not passed to an *embed_** function, this global connection will be used.

Parameters

- **auth_key** (*str*) – Basilica API key. You can view your auth keys at https://basilica.ai/auth_keys.
- **server** (*str*) – Basilica server to point to (Default: *https://api.basilica.ai*)

```
>>> conn <- connect("SLOW_DEMO_KEY") # Create a connection to pass to functions
embeddings <- embed_sentences(c("hello world"), conn=conn)
```

```
>>> connect("SLOW_DEMO_KEY") # Populate the global connection
embeddings <- embed_sentences(c("hello world"))
embeddings <- embed_sentences(c("hello world")) # Will both use the sameglobal␣
↪connection
```

basilica.**embed_sentence**(*sentence*, *model*, *version*, *conn*, *timeout*)

Get a vector of features for a sentence

Parameters

- **sentence** – Sentence or string
- **model** (*character()*) – Name of the image model you wish to use. (Default: *english*)
- **version** (*character()*) – Version of the image model you wish to use. (Default: *default*)
- **conn** (*environment()*) – Basilica connection. Must be created with the *connect* function (Default: Global *basilica_connection*)
- **timeout** (*number()*) – Time (in seconds) before requests times out. (Default *5*)

Returns An embedding.

Return type Matrix

basilica.**embed_sentences**(*sentences*, *model*, *version*, *conn*, *timeout*)
Get a vector of features for a list of sentences

Parameters

- **sentence** – Sentence or string
- **model** (*character()*) – Name of the image model you wish to use. (Default: *english*)
- **version** (*character()*) – Version of the image model you wish to use. (Default: *default*)
- **conn** (*environment()*) – Basilica connection. Must be created with the *connect* function (Default: Global *basilica_connection*)
- **timeout** (*number()*) – Time (in seconds) before requests times out. (Default *5*)

Returns An embedding.

Return type Matrix

basilica.**embed_image**(*image*, *model*, *version*, *conn*, *timeout*)
Get a vector of features for an image

Parameters

- **image** (*raw()*) – Raw vector read from image file (JPEG or PNG)
- **model** (*character()*) – Name of the image model you wish to use. (Default: *generic*)
- **version** (*character()*) – Version of the image model you wish to use. (Default: *default*)
- **conn** (*environment()*) – Basilica connection. Must be created with the *connect* function (Default: Global *basilica_connection*)
- **timeout** (*number()*) – Time (in seconds) before requests times out. (Default *5*)

Returns An embedding.

Return type Matrix

basilica.**embed_image_file**(*image_path*, *model*, *version*, *conn*, *timeout*)
Get a vector of features for an image

Parameters

- **image_path** – Path to an image (JPEG or PNG)
- **model** (*character()*) – Name of the image model you wish to use. (Default: *generic*)
- **version** (*character()*) – Version of the image model you wish to use. (Default: *default*)
- **conn** (*environment()*) – Basilica connection. Must be created with the *connect* function (Default: Global *basilica_connection*)
- **timeout** (*number()*) – Time (in seconds) before requests times out. (Default *5*)

Returns An embedding.

Return type Matrix

basilica.**embed_image_files**(*image_paths*, *model*, *version*, *conn*, *timeout*)
Get a vector of features for a list images

Parameters

- **image_paths** – List of file paths to images (JPEG or PNG)
- **model** (*character()*) – Name of the image model you wish to use. (Default: *generic*)
- **version** (*character()*) – Version of the image model you wish to use. (Default: *default*)
- **conn** (*environment()*) – Basilica connection. Must be created with the *connect* function (Default: Global *basilica_connection*)
- **timeout** (*number()*) – Time (in seconds) before requests times out. (Default *5*)

Returns An embedding.

Return type Matrix

basilica.**embed_images**(*images*, *model*, *version*, *conn*, *timeout*)

Get a vector of features for a list images

Parameters

- **images** (*list()*) – List of raw vectors read from image files (JPEG or PNG)
- **model** (*character()*) – Name of the image model you wish to use. (Default: *generic*)
- **version** (*character()*) – Version of the image model you wish to use. (Default: *default*)
- **conn** (*environment()*) – Basilica connection. Must be created with the *connect* function (Default: Global *basilica_connection*)
- **timeout** (*number()*) – Time (in seconds) before requests times out. (Default *5*)

Returns An embedding.

Return type Matrix

CHAPTER 3

Vignettes

3.1 Working with images

Basilica provides 4 functions for working with images:

- `embed_image`
- `embed_images`
- `embed_image_file`
- `embed_image_files`

The `embed_image_file` and `embed_image_files` functions take a `characther` vector (a string) with a file path pointing to an image. On the other hand, `embed_image` and `embed_images` take a `raw` vector obtained through `readBin`.

3.1.1 `embed_image_file`

```
download.file("https://www.basilica.ai/static/images/tutorial/dog-test-1.jpg", "/tmp/
↪dog1.jpg")
download.file("https://www.basilica.ai/static/images/tutorial/dog-test-2.jpg", "/tmp/
↪dog2.jpg")
download.file("https://www.basilica.ai/static/images/tutorial/cat-test-1.jpg", "/tmp/
↪cat.jpg")

library('basilica')
conn <- connect("SLOW_DEMO_KEY")

embeddings = list()
embeddings[[1]] = embed_image_file("/tmp/dog1.jpg", conn=conn)
embeddings[[2]] = embed_image_file("/tmp/dog2.jpg", conn=conn)
embeddings[[3]] = embed_image_file("/tmp/cat.jpg", conn=conn)
```

(continues on next page)

(continued from previous page)

```
print(cor(embeddings[[1]], embeddings[[2]]))
print(cor(embeddings[[1]], embeddings[[3]]))
```

3.1.2 embed_image_files

```
download.file("https://www.basilica.ai/static/images/tutorial/dog-test-1.jpg", "/tmp/
↪dog1.jpg")
download.file("https://www.basilica.ai/static/images/tutorial/dog-test-2.jpg", "/tmp/
↪dog2.jpg")
download.file("https://www.basilica.ai/static/images/tutorial/cat-test-1.jpg", "/tmp/
↪cat.jpg")


library('basilica')
conn <- connect("SLOW_DEMO_KEY")

embeddings = embed_image_files(c("/tmp/dog1.jpg", "/tmp/dog2.jpg", "/tmp/cat.jpg"),␣
↪conn=conn)

print(cor(embeddings[1,], embeddings[2,]))
print(cor(embeddings[1,], embeddings[3,]))
```

3.1.3 embed_image

```
download.file("https://www.basilica.ai/static/images/tutorial/dog-test-1.jpg", "/tmp/
↪dog1.jpg")
download.file("https://www.basilica.ai/static/images/tutorial/dog-test-2.jpg", "/tmp/
↪dog2.jpg")
download.file("https://www.basilica.ai/static/images/tutorial/cat-test-1.jpg", "/tmp/
↪cat.jpg")

library('basilica')
conn <- connect("SLOW_DEMO_KEY")

embeddings = list()

f <- file("/tmp/dog1.jpg", "rb")
dog1_raw <- readBin(f, "raw", file.info("/tmp/dog1.jpg")[1, "size"])
close(f)
embeddings[[1]] = embed_image(dog1_raw, conn=conn)

f <- file("/tmp/dog2.jpg", "rb")
dog2_raw <- readBin(f, "raw", file.info("/tmp/dog2.jpg")[1, "size"])
close(f)
embeddings[[2]] = embed_image(dog2_raw, conn=conn)

f <- file("/tmp/cat.jpg", "rb")
cat_raw <- readBin(f, "raw", file.info("/tmp/cat.jpg")[1, "size"])
close(f)
embeddings[[3]] = embed_image(cat_raw, conn=conn)

print(cor(embeddings[[1]], embeddings[[2]]))
print(cor(embeddings[[1]], embeddings[[3]]))
```

3.1.4 embed_images

```
download.file("https://www.basilica.ai/static/images/tutorial/dog-test-1.jpg", "/tmp/
↪dog1.jpg")
download.file("https://www.basilica.ai/static/images/tutorial/dog-test-2.jpg", "/tmp/
↪dog2.jpg")
download.file("https://www.basilica.ai/static/images/tutorial/cat-test-1.jpg", "/tmp/
↪cat.jpg")

library('basilica')
conn <- connect("SLOW_DEMO_KEY")

embeddings = list()

f <- file("/tmp/dog1.jpg", "rb")
dog1_raw <- readBin(f, "raw", file.info("/tmp/dog1.jpg")[1, "size"])
close(f)

f <- file("/tmp/dog2.jpg", "rb")
dog2_raw <- readBin(f, "raw", file.info("/tmp/dog2.jpg")[1, "size"])
close(f)

f <- file("/tmp/cat.jpg", "rb")
cat_raw <- readBin(f, "raw", file.info("/tmp/cat.jpg")[1, "size"])
close(f)

embeddings = embed_images(list(dog1_raw, dog2_raw, cat_raw), conn=conn)

print(cor(embeddings[1,], embeddings[2,]))
print(cor(embeddings[1,], embeddings[3,]))
```

3.2 Working with text

Basilica provides 2 functions for working with images:

- `embed_sentence`
- `embed_sentences`

The `embed_sentence` function takes a single `characther` vector (a string) and returns a vector of features. The `embed_sentences` functions takes a list of `character` vectors returns a matrix with a feature vector on every row.

3.2.1 embed_sentece

```
sentences <- c(
  "This is a sentence!",
  "This is a similar sentence!",
  "I don't think this sentence is very similar at all..."
)

library('basilica')
conn <- connect("SLOW_DEMO_KEY")
```

(continues on next page)

(continued from previous page)

```
embeddings <- list()
embeddings[[1]] <- embed_sentence(sentence[[1]], conn=conn)
embeddings[[2]] <- embed_sentence(sentence[[2]], conn=conn)
embeddings[[3]] <- embed_sentence(sentence[[3]], conn=conn)

print(cor(embeddings[[1]], embeddings[[2]]))
print(cor(embeddings[[1]], embeddings[[3]]))
```

3.2.2 embed_senteces

```
sentences <- c(
  "This is a sentence!",
  "This is a similar sentence!",
  "I don't think this sentence is very similar at all..."
)

library('basilica')
conn <- connect("SLOW_DEMO_KEY")

embeddings <- embed_sentences(sentences, conn=conn)

print(cor(embeddings[1,], embeddings[2,]))
print(cor(embeddings[1,], embeddings[3,]))
```

3.3 Training a logistic regression from Twitter comments

In this example, we'll train a logistic regression to classify tweets using only the natural language text found in these tweets. We'll only need about 800 tweets per user account.

3.3.1 Setup

To run this example, you will need the following packages.

```
install.packages("dplyr", "ROCR")
```

3.3.2 Step 1: Embedding all tweets

We'll use a collection of about 800 tweets from Bill Gates and Kanye West and train a logistic regression to predict (given a tweet) which account the tweet belongs to. In order to do that, we'll first load the tweets from the basilica package.

```
library(jsonlite)
bill <- fromJSON(system.file("extdata/twitter/billgates.json", package="basilica"))
kanye <- fromJSON(system.file("extdata/twitter/kanyewest.json", package="basilica"))
```

Now that we've loaded the JSON files, we can embedded the text of these tweets using Basilica.

```
library(basilica)
conn <- connect("05e19f1c-39de-ed9c-ae42-feab42f5f84d")

embeddings <- rbind(embed_sentences(bill[, 7], conn=conn), embed_sentences(kanye[, 7],
↪ conn=conn)) # 7 is the index of the text
```

3.3.3 Step 2: Running PCA + Cleaning Data

Now that we have these embeddings, we'll want to run PCA and get the 100 features that explain the most variance. We'll also add a column to the matrix with the corresponding category each tweet belongs to.

```
pca <- prcomp(t(embeddings), center = TRUE,scale = TRUE)
features <- pca$rotation[,1:100]

type <- c(integer(dim(bill)[1]) + 1, integer(dim(kanye)[1]))
features <- cbind(type, features)
features <- data.frame(features[sample.int(nrow(features)),])
```

3.3.4 Step 3: Training the model

Finally, we can now train our model. In order to do that we'll separate out the data into training and test data.

```
library(dplyr)
train_data <- sample_frac(features, 0.8)
train_index <- as.numeric(rownames(train_data))
test_data <- features[-train_index, ]

model <- glm(type ~ ., data = train_data, family = "binomial")
```

3.3.5 Step 4: Verifying Results

After training the model, we can verify who well it's trained by taking a look at the confusion matrix.

```
predict <- predict(model, newdata=test_data, type = 'response')
table(train_data$type, predict > 0.5)

library(ROCR)
ROCRpred <- prediction(predict, test_data$type)
ROCRperf <- performance(ROCRpred, 'tpr','fpr')
plot(ROCRperf, colorize = TRUE, text.adj = c(-0.2,1.7))
```

You have now trained a logistic regression with only the natural language text of the tweets and 800 data points per category and getting an R squared of about `0.80`.

CHAPTER 4

Basilica R Client: Deep Feature Extraction for Images and Text

4.1 Word2Vec For Anything

Basilica allows you to easily augment your models with images and text. You send us an image or a snippet of natural language text and we send you a vector of features you can use to train your models.

4.2 Installation

You can install the released version of this package from Google cloud:

```
install.packages("https://storage.googleapis.com/basilica-r-client/basilica_0.0.2.tar.
↪gz", repos=NULL)
```

or from Github (requires the `devtools` package):

```
devtools::install_github("basilica-ai/basilica-r-client")
```

(CRAN submission approval in progress)

4.3 Examples

This is a basic example which shows you how to solve a common problem:

4.3.1 Creating a Connection

Before embedding an image or text (getting a vector of features), you must first connect to the API with a demo key. `SLOW_DEMO_KEY` is a key you can use for testing with a low per-week limit, but you can create API keys for free at www.basilica.ai.

```
library('basilica')
# Create a connection
# You can use our `SLOW_DEMO_KEY` (it actually works) or create your own at basilica.
↪ai
conn <- connect("SLOW_DEMO_KEY")
```

4.3.2 Embedding Text

Getting a vector of features for text:

```
sentences = c(
    "This is a sentence!",
    "This is a similar sentence!",
    "I don't think this sentence is very similar at all..."
)

# Returns a data frame with 512 features for each of the 3 sentences
embeddings <- embed_sentences(sentences, conn=conn)
print(dim(embeddings)) # 3 512
print(embeddings) # [[0.8556405305862427, ...], ...]

print(cor(embeddings[1,], embeddings[2,])) # 0.8048559
print(cor(embeddings[1,], embeddings[3,])) # 0.6877435
```

Differences from Word2Vec

It's important to know that the embedding you get for a sentence is completely different from an embedding you would get with Word2Vec. Word2Vec returns a word-level embedding, while basilica is trained on longer snippets of natural language text (phrases, sentences, paragraphs). For that reason, results on models where the context of the sentence matter (like sentiment analysis) will get much better results with a sentence-level embedding than with a word embedding.

4.3.3 Embedding an Image

Getting a vector of features for images:

```
embeddings <- embed_image("/tmp/image.jpg", conn=conn)
print(dim(embeddings)) # 1 2048
print(embeddings) # [[0.8556405305862427, ...], ...]
```

4.4 Development

If you want to contribute to this client, here's are some of the libraries and commands you will need:

4.4.1 Setup

```
brew install qpdf
```

```
install.packages("devtools")
install.packages("usethis")
install.packages("testthat")
```

4.4.2 Building

When on a branch, make sure all these commands work and pass.

```
devtools::test()
devtools::document()
devtools::build_vignettes()
devtools::check()
```

Python Module Index

b

Index

B

C

E