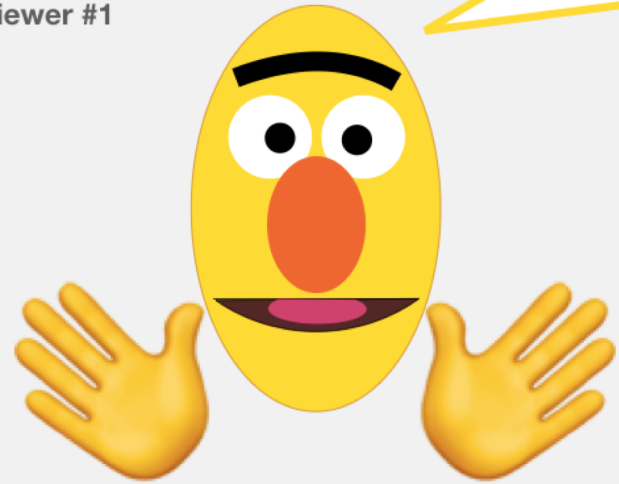
A Visual Guide to Using BERT for the First Time

Adapted from: http://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

"a visually stunning rumination on love"

Reviewer #1

That's a positive thing to say



"reassembled from the cutting room floor of any given daytime soap"

Reviewer #2

That's negative

A Guide to Using BERT on Colab

- This tutorial is an adaptation of: http://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/ for how to use a variant of BERT to classify sentences.

- This is an example that is basic enough as a first intro, yet advanced enough to showcase some of the key concepts involved.

- There is an associated iPython Notebook that can be run directly in Google Colab.

Dataset: SST2

Dataset used in example is SST2 --- contains sentences from movie reviews, each labeled as either positive (has the value 1) or negative (has the value 0):

sentence	label
a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films	1
apparently reassembled from the cutting room floor of any given daytime soap	0
they presume their audience won't sit still for a sociology lesson	0
this is a visually stunning rumination on love , memory , history and the war between art and commerce	1
jonathan parker 's bartleby should have been the be all end all of the modern office anomie films	1

Models: Sentence Sentiment Classification

Goal: Create model that takes a sentence and classifies it as either “positive” (1) or “negative” (0) sentiment:

The BERT model for text classification has 2 parts:

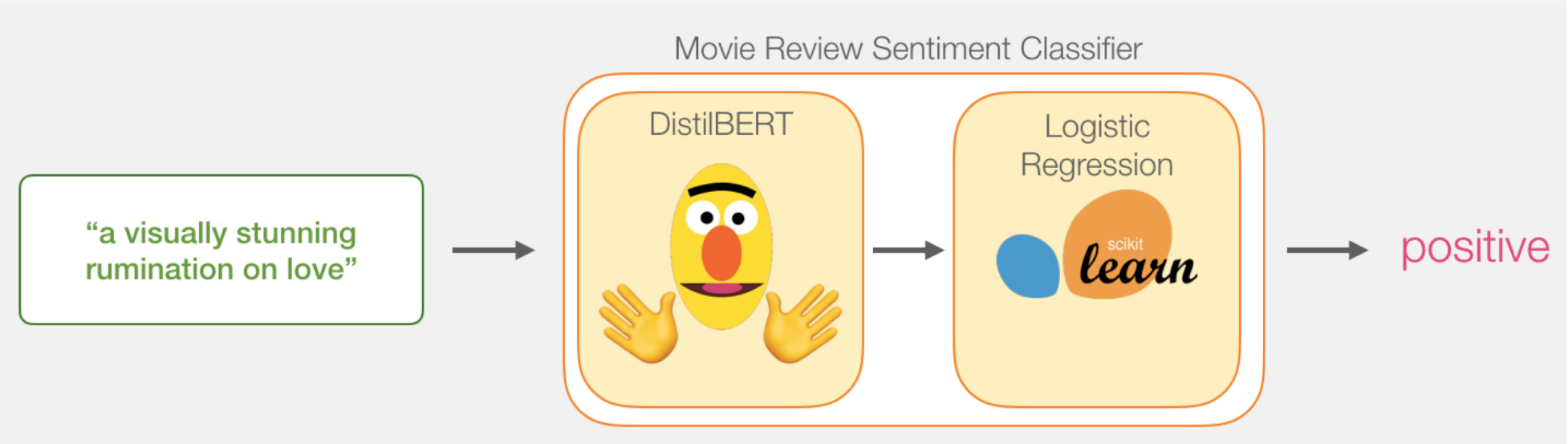
(1) DistilBERT processes the text and passes along some information it extracted from the text to the next model.
   - DistilBERT is a smaller version of BERT developed and open sourced by the team at HuggingFace. It's a lighter and faster version of BERT that roughly matches its performance.

(1) The next model, a basic Logistic Regression model from scikit learn, takes the result from Step (1) and classify the text as either positive or negative (1 or 0, respectively).

Data passed between the 2 models is a vector of size 768.

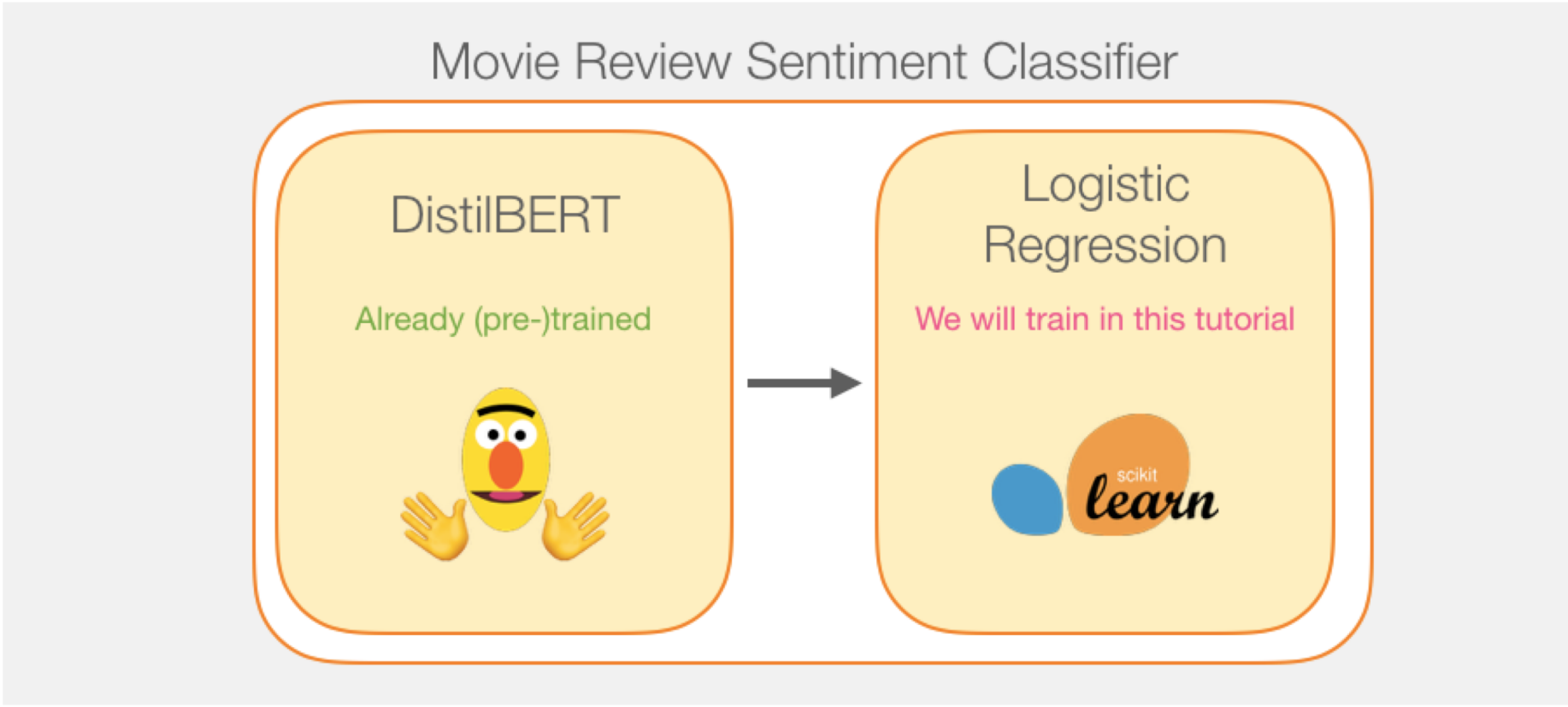
This of vector is an embedding of the text that is used for classification.

The 2-stage model for classification of text



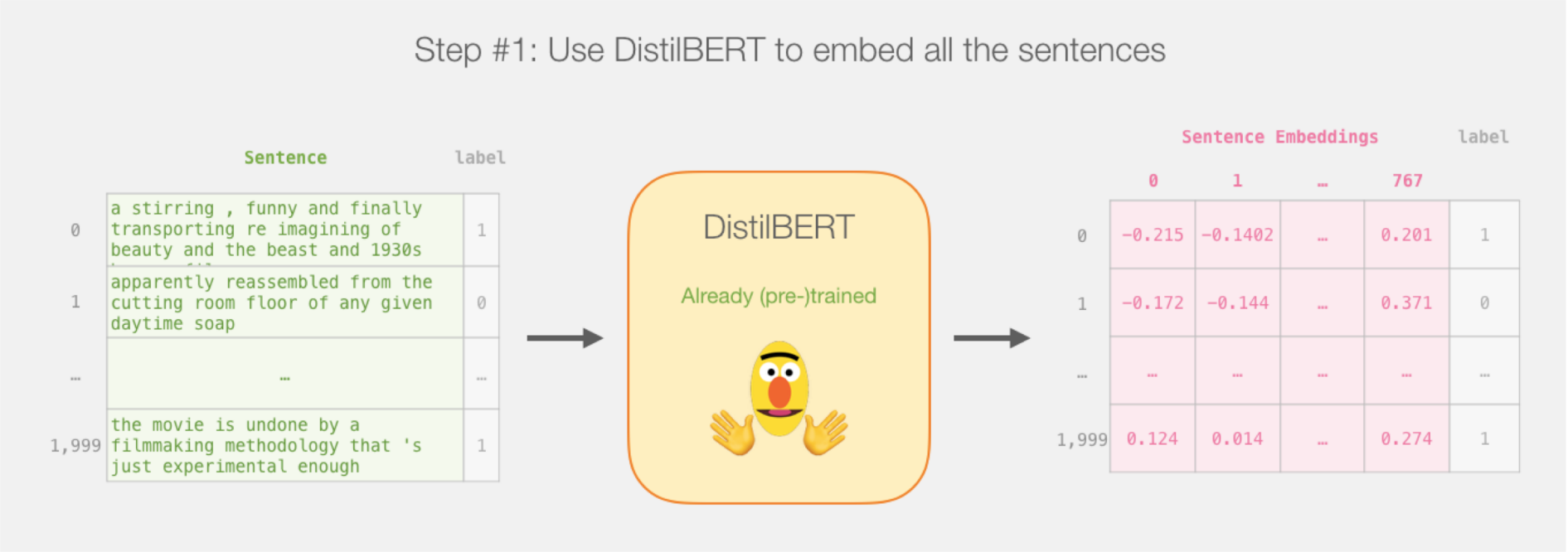
Model Training

- BERT is a “Transfer Learning” Model:
  - The 1st-stage model is already pre-trained on a very large dataset and to model the English language. But, BERT neither trained nor fine-tuned for a specific text classification task (such as the sentiment of the text).
- 2nd-stage: Use a logistic regression that takes the summarized text properties (called ***embeddings***) captured by BERT model and then correlates the properties with a text “label” (such as sentiment).
- Only train the logistic regression model using labelled text observations (either positive (1) or negative (0) sentiment)

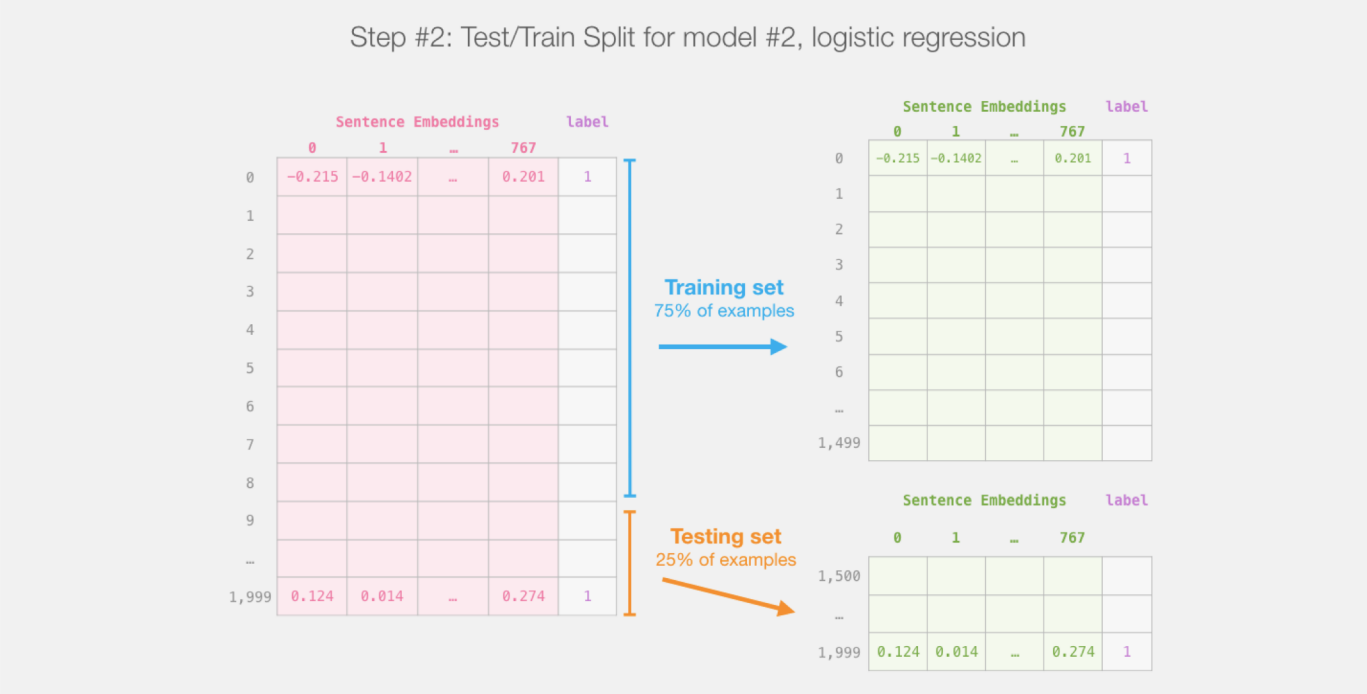

Movie Review Sentiment Classifier
DistilBERT
Already (pre-)trained
Logistic Regression
We will train in this tutorial
scikit
learn

Tutorial Overview

Step #1: Use trained distilBERT to generate sentence embeddings for 2,000 sentences.

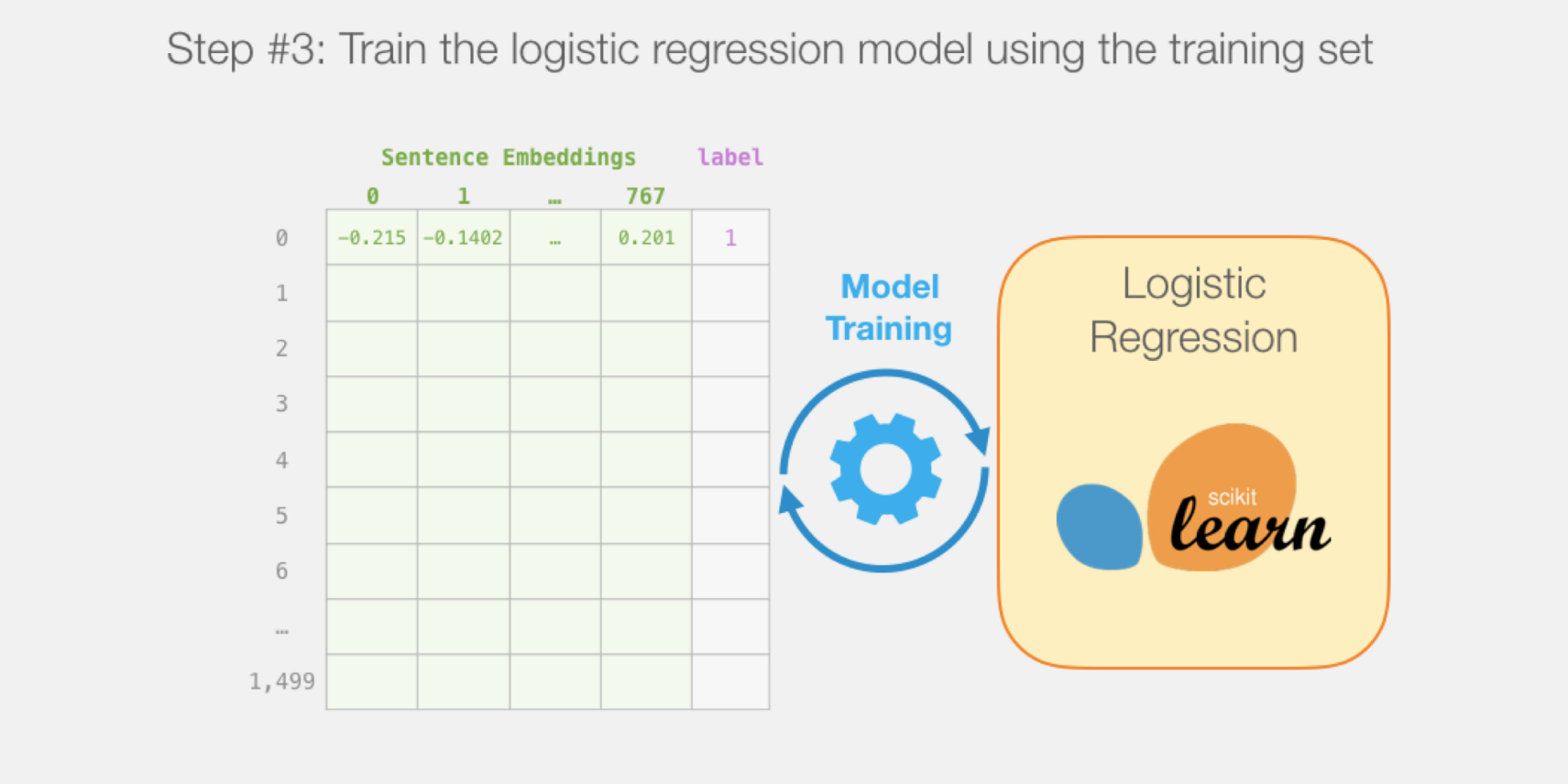


Step #2: We will not touch distilBERT after this Step #1. Then, we use Scikit Learn to do a train/test split on this dataset:



Train/test split for output of *DistilBert* (Model #1) creates the dataset for the *Logistic regression* (Model #2). Sklearn's train/test split shuffles observations before making the split (it doesn't just take first 75% observations in dataset).

Then train the Logistic regression model on the training set:

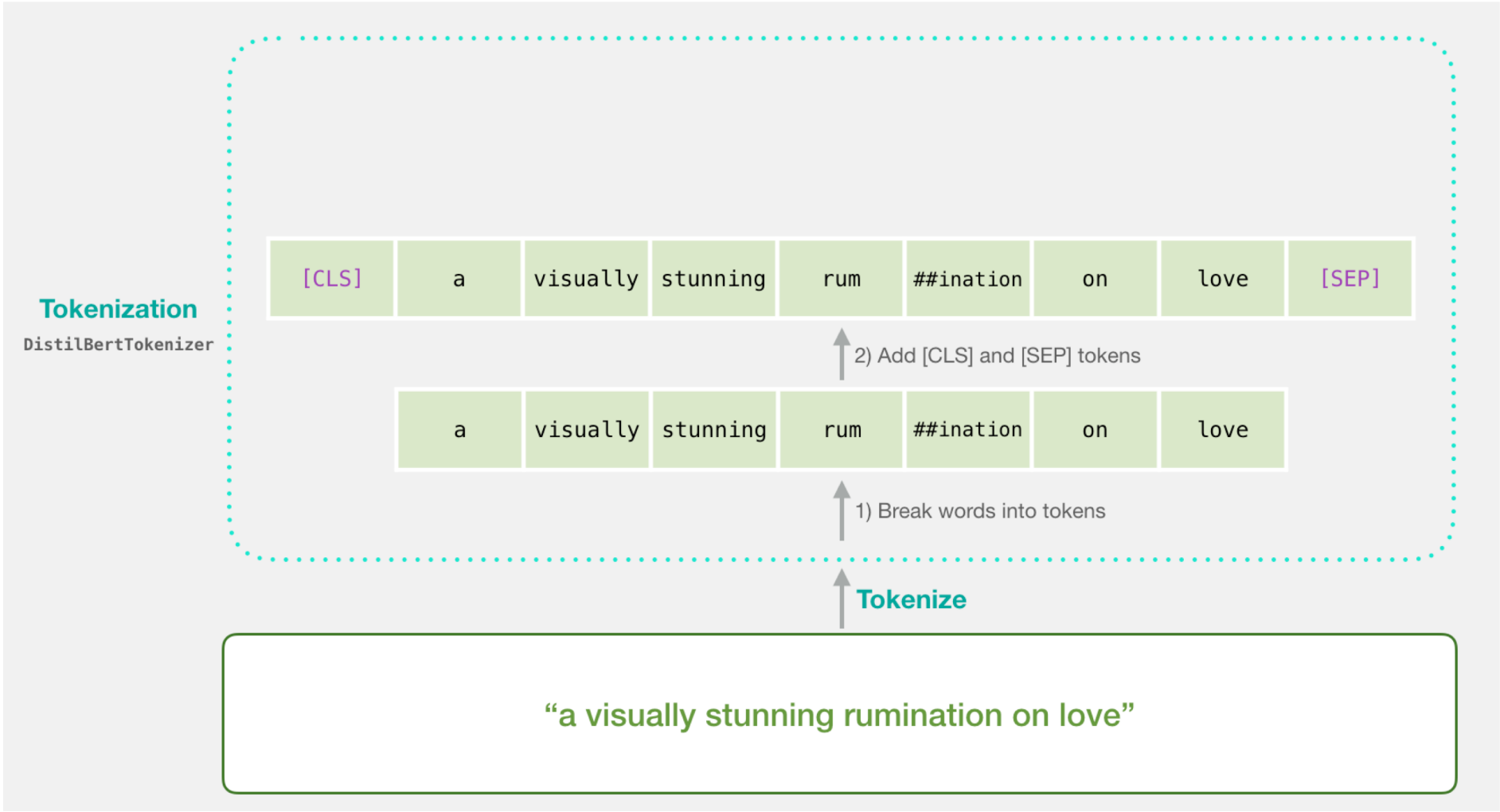


How a single prediction is calculated

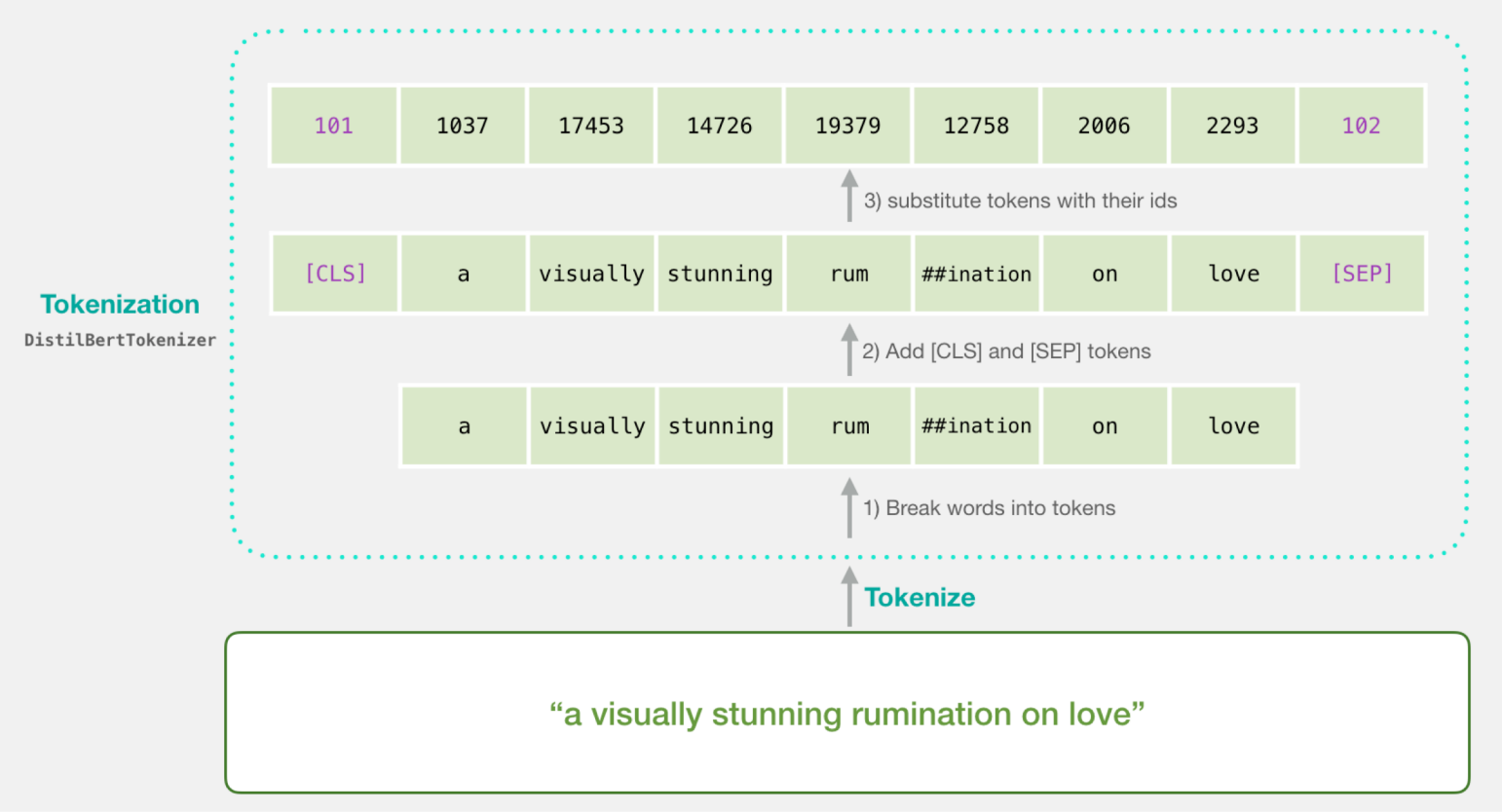
How a trained model calculates its prediction:

Example: Classify the sentence “a visually stunning rumination on love”.

- The first step is to use the BERT tokenizer to first split the text into tokens.
- Then, add the special tokens needed for text classification:
  - [CLS] at the first position (the classification vector of 768 numbers)
  - [SEP] at the end of the text (a separator).

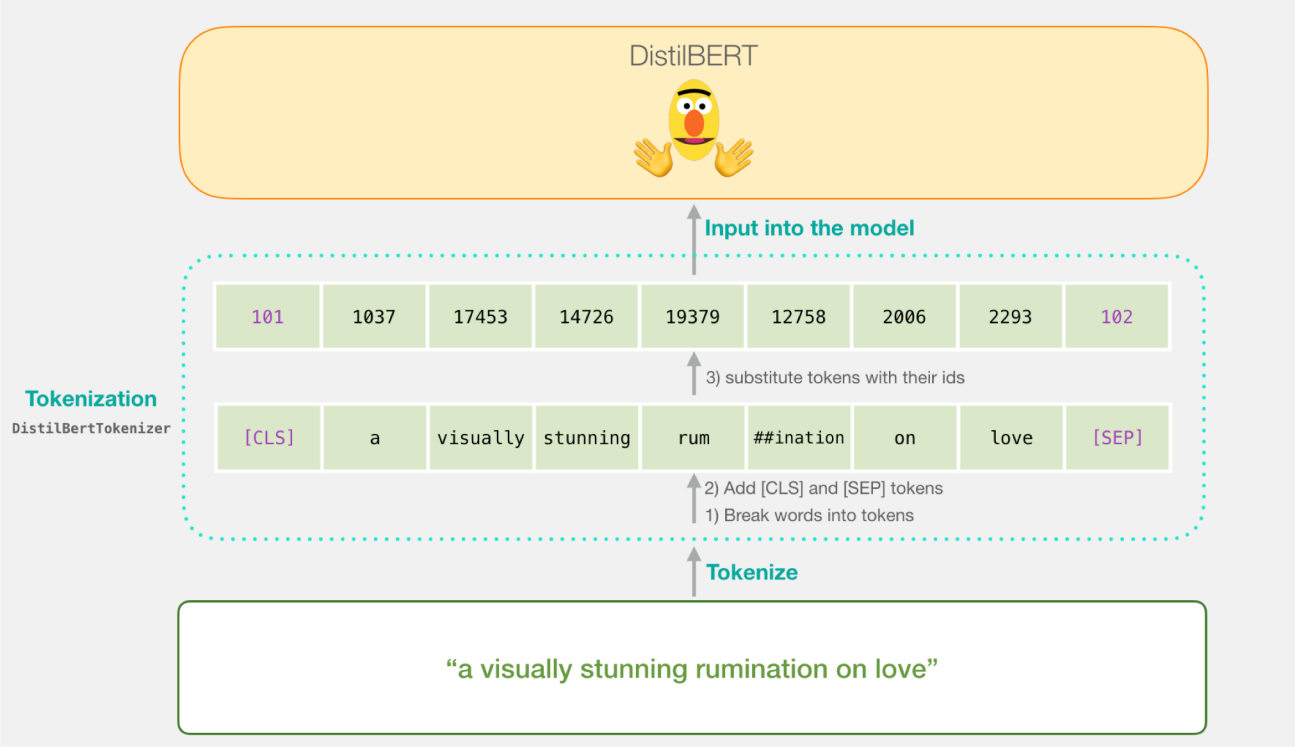

Tokenization
DistilBertTokenizer
[CLS]
a
visually
stunning
rum
##ination
on
love
[SEP]
2) Add [CLS] and [SEP] tokens
a
visually
stunning
rum
##ination
on
love
1) Break words into tokens
Tokenize
“a visually stunning rumination on love”

The tokenizer replaces each token with its id from the “embedding table” which is a component we get with the trained BERT model.



The tokenizer does all these steps in a single line of code:

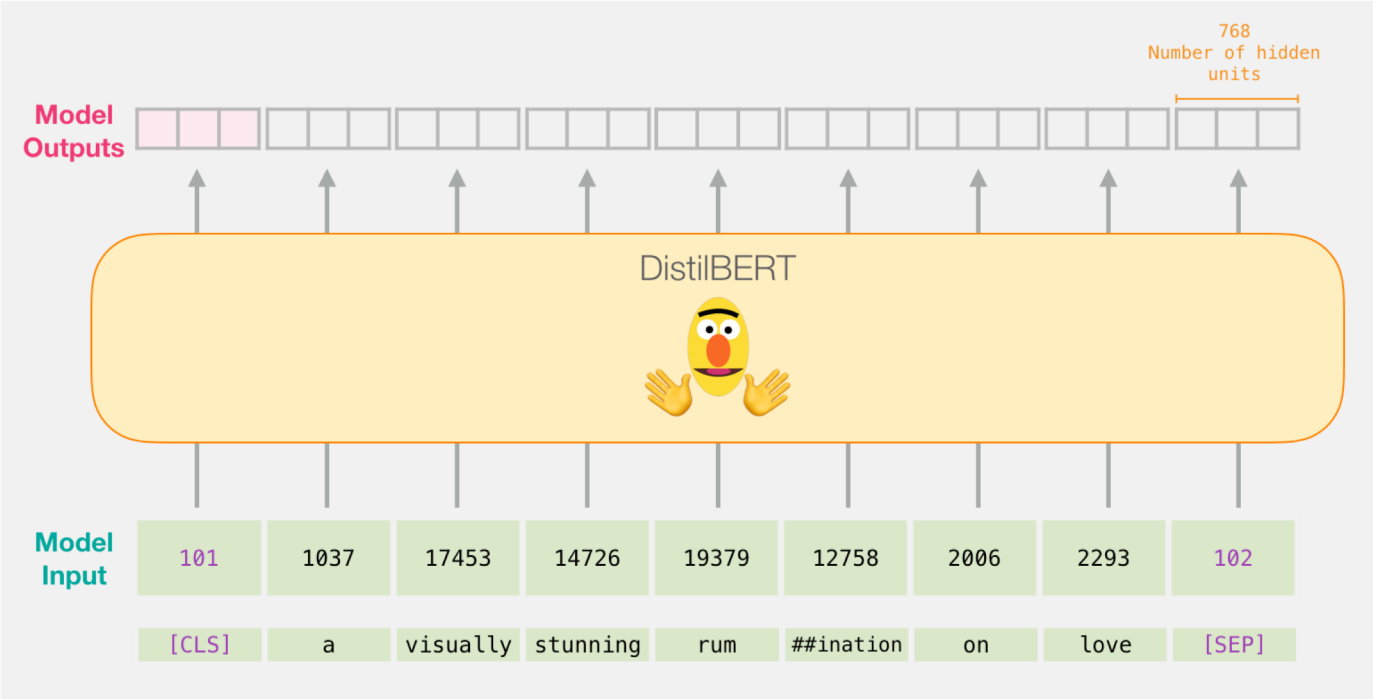
```
tokenizer.encode("a visually stunning rumination on love", add_special_tokens=True)
```



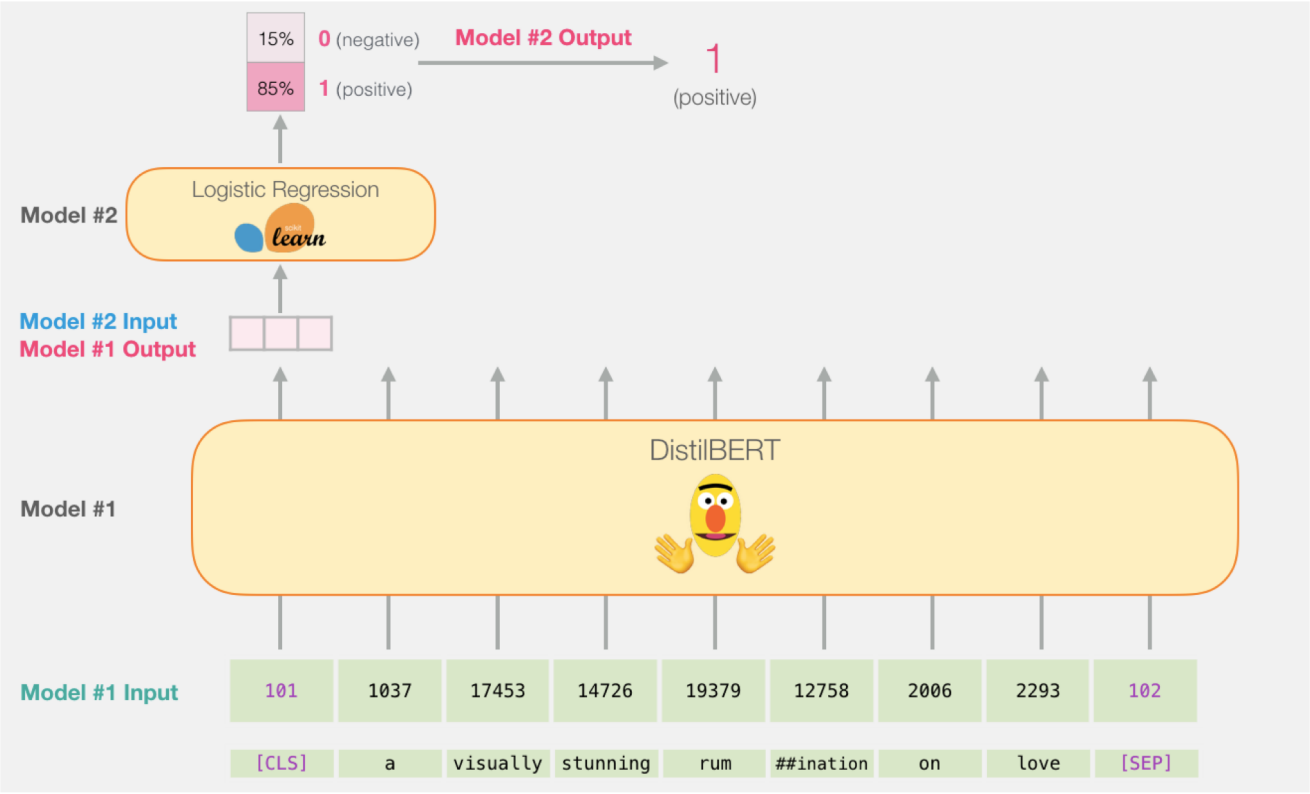
The input sentence is now the proper shape to be passed to DistilBERT.

Flowing Through DistilBERT

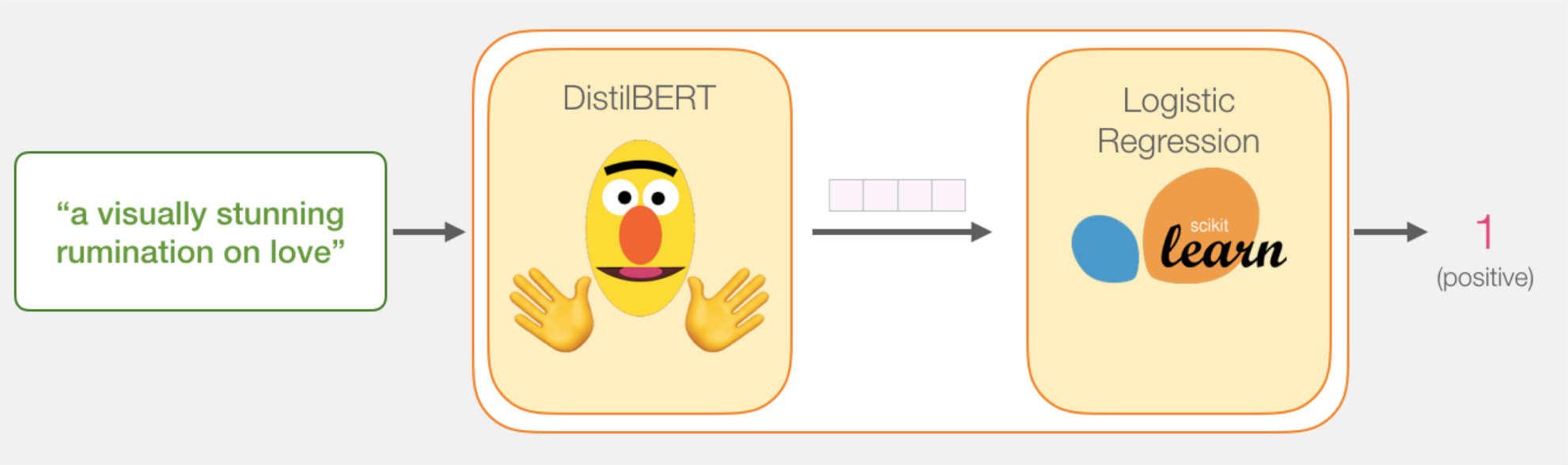
The input vector is then passed through *DistilBERT* and the output is a vector for each input token. Each vector is made up of **768 numbers**.



Because this is a text classification task, we ignore all except the first vector (the one associated with the [CLS] token).
This [CLS] vector is used as the input to the Logistic regression model.



The Logistic regression model uses this [CLS] vector to predict the label (1=positive, 0=negative):



The Notebook Python Code

The notebook containing all this code is available on colab and github.
The first code box importing the tools of the necessary libraries:

```
import numpy as np
import pandas as pd
import torch
import transformers as ppb # pytorch transformers
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

The dataset is available as a file on github, so we just import it directly into a pandas dataframe

```
df = pd.read_csv('https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/SST2/train.tsv', delimiter='\t', header=None)
```

We can use df.head() to look at the first five rows of the dataframe to see how the data looks.

```
df.head()
```

The df.head() command create the output on the next slide:

	0	1
0	a stirring , funny and finally transporting re...	1
1	apparently reassembled from the cutting room f...	0
2	they presume their audience wo n't sit still f...	0
3	this is a visually stunning rumination on love...	1
4	jonathan parker 's bartleby should have been t...	1

Importing pre-trained DistilBERT model and tokenizer

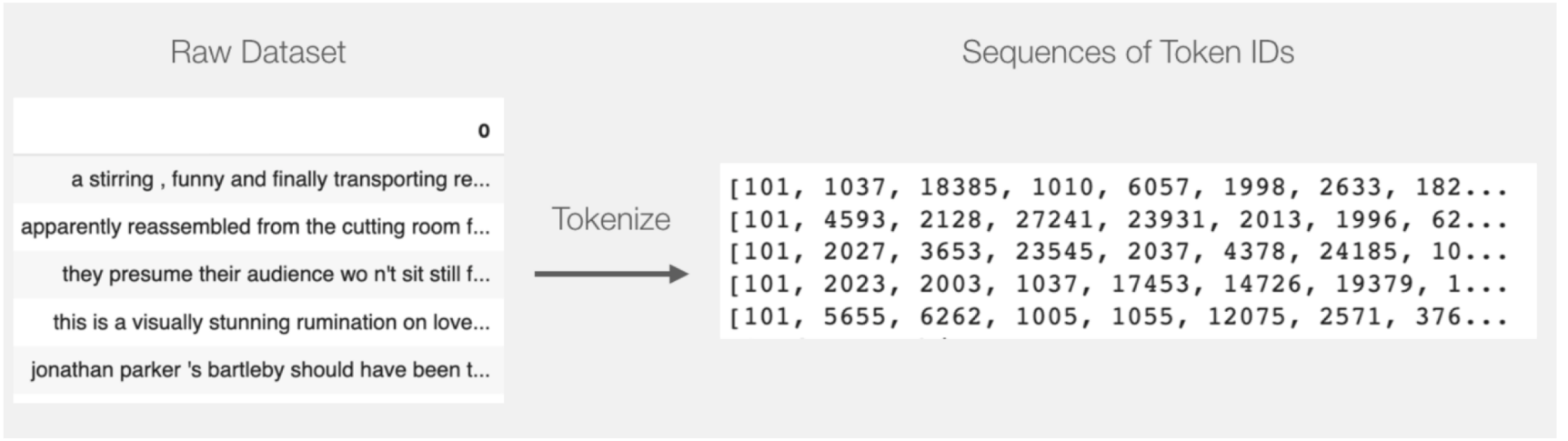
```
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.DistilBertTokenizer
, 'distilbert-base-uncased')

## Want BERT instead of distilBERT? Uncomment the following line:
#model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ppb.BertTokenizer, 'bert-bas
e-uncased')

# Load pretrained model/tokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

Now tokenize the dataset --- This code will tokenize and process all text observations together as a batch (in this case, 2000 observations).

Tokenization

```
tokenized = df[0].apply((lambda x: tokenizer.encode(x, add_special_tokens=True)))
```

This turns every sentence into the list of ids.



- Dataset is currently a list (or Pandas DataFrame) of lists.
- Before DistilBERT can process this as an input, must make all the input vectors the same size by padding shorter sentences with the token id 0.
- After the padding, we have a matrix/tensor that is ready to be passed to BERT:

BERT/DistilBERT Input Tensor

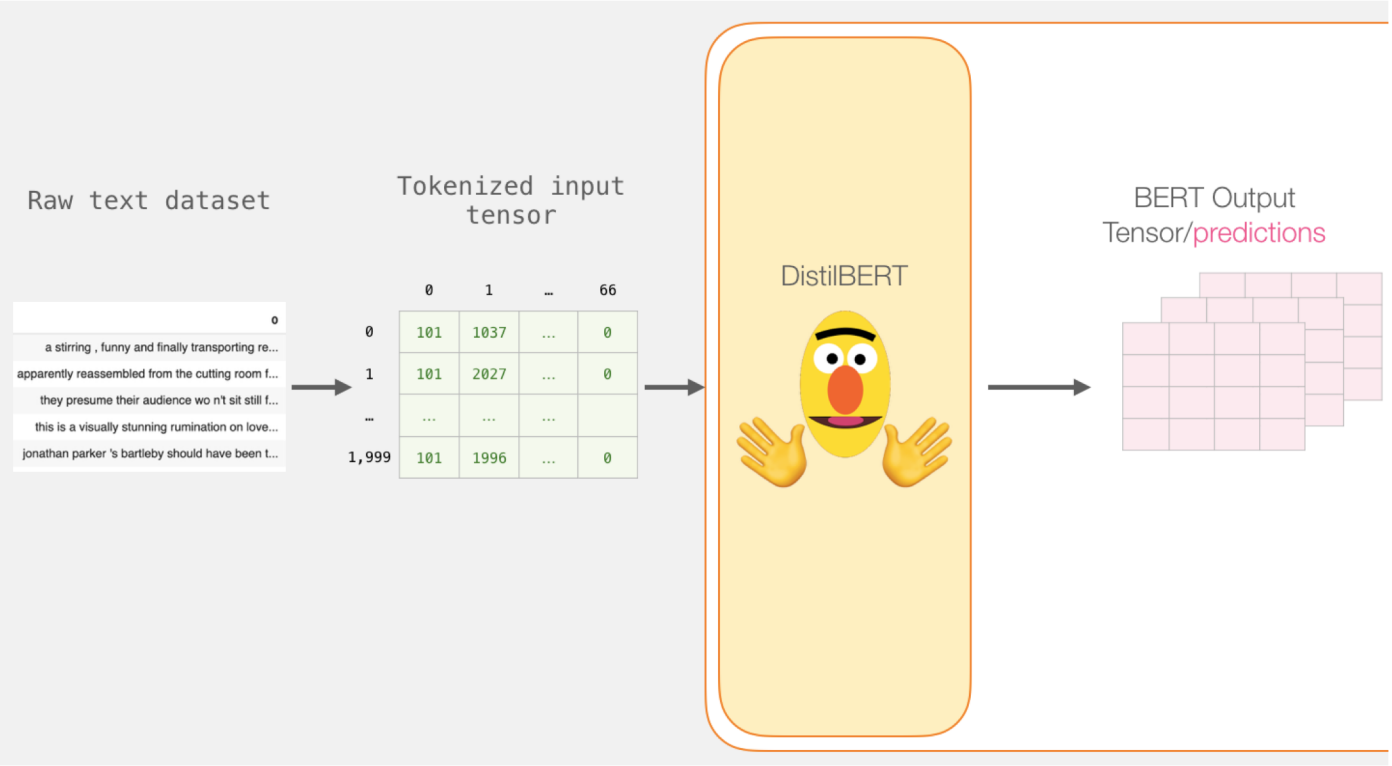
Input sequences (reviews)	Tokens in each sequence: 0	1	…	66
0	101	1037	…	0
1	101	2027	…	0
…	…	…	…	
1,999	101	1996	…	0

Processing with DistilBERT

Then create an input tensor using the padded token matrix, and send it to DistilBERT:

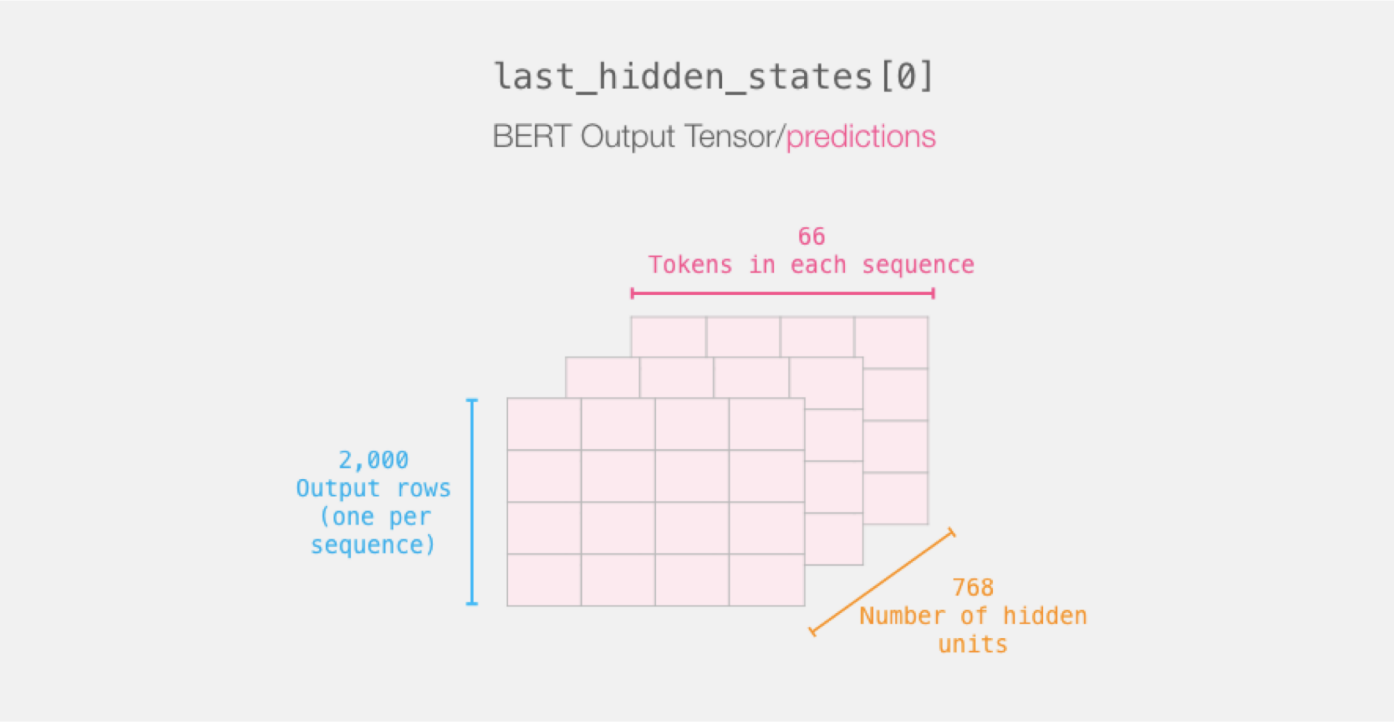
```
input_ids = torch.tensor(np.array(padded))

with torch.no_grad():
    last_hidden_states = model(input_ids)
```

- After running this step, `last_hidden_states` holds the outputs of DistilBERT.
- It is a tuple with the shape: (number of examples, max number of tokens in the sequence, number of hidden units in the DistilBERT model).
- In this example, tuple is: 2000 (there are 2000 observations), 66 (which is the number of tokens in the longest text sequence from the 2000 examples), 768 (the number of hidden units in the DistilBERT model).



Unpacking the BERT output tensor

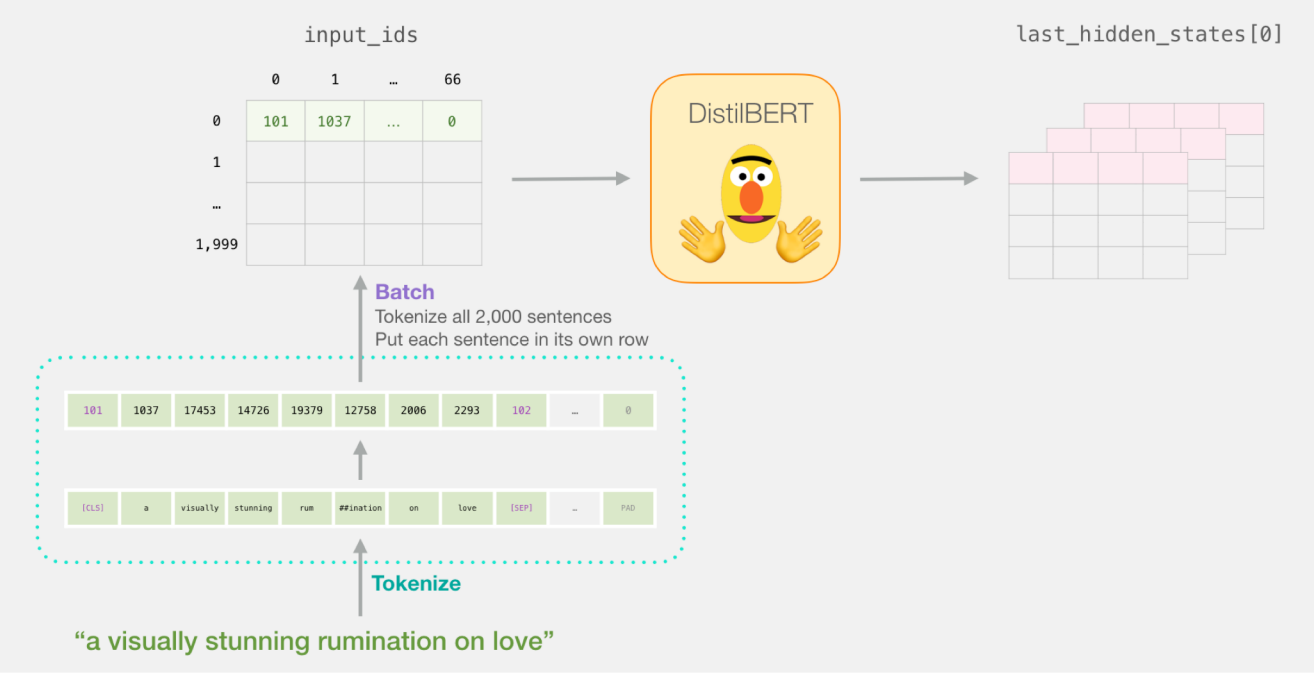
Unpack this 3-D output tensor: First start by examining its dimensions:



Recapping a sentence's journey

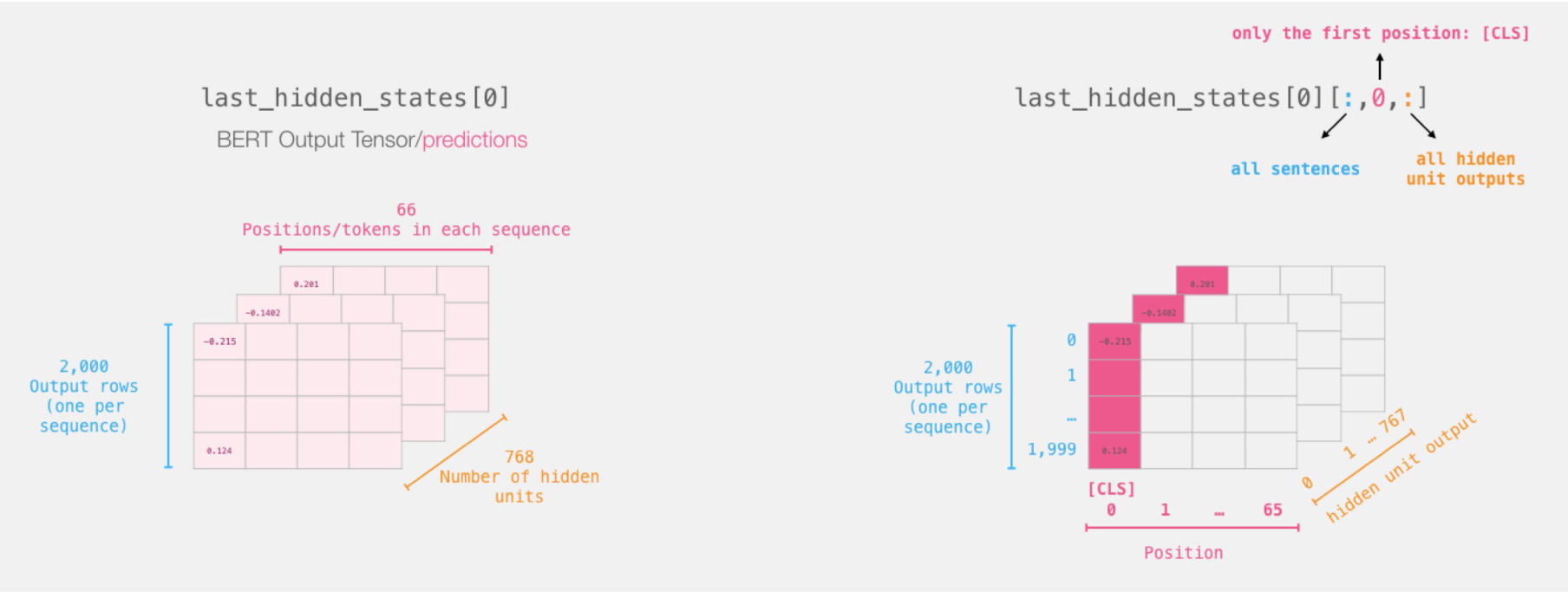
Each row is associated with a text observation (sentence) from the dataset.

To recap the processing path of the first sentence,think of it as looking like this:



Slicing the important part

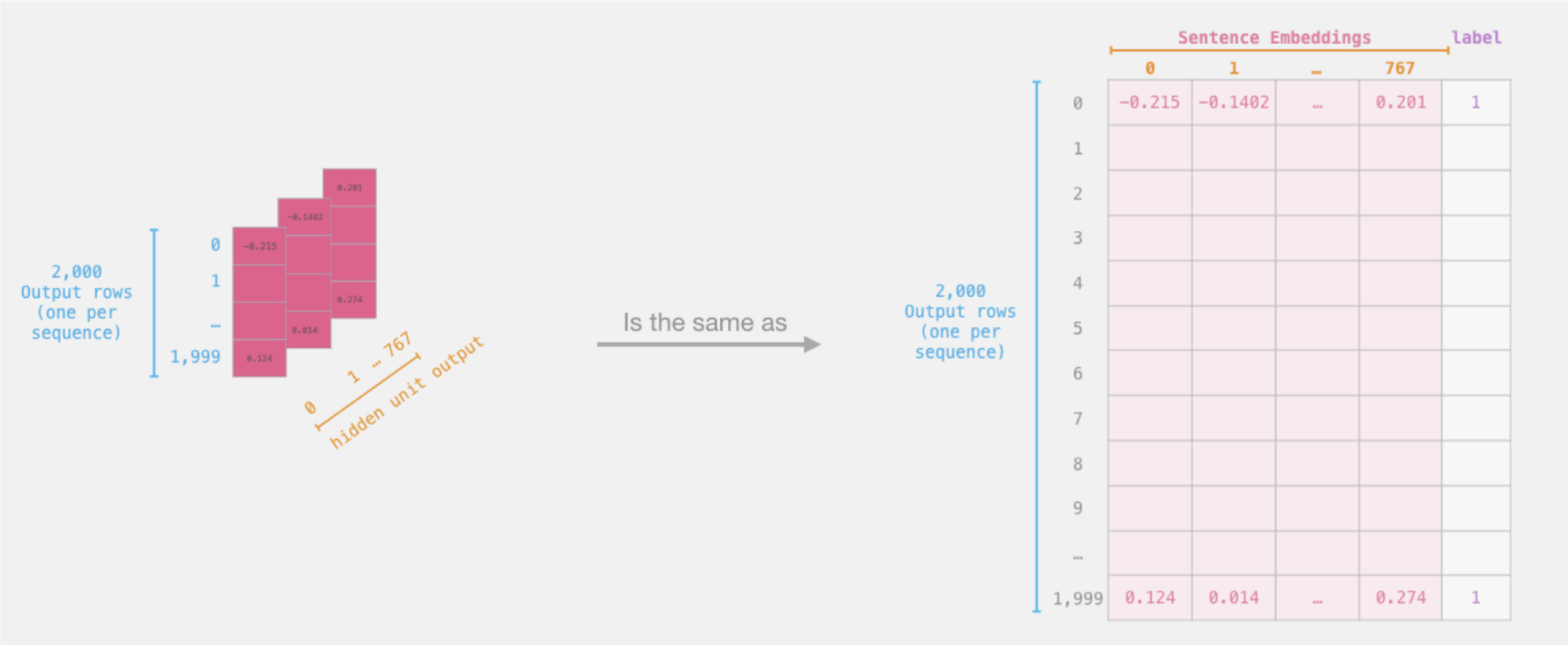
For text classification, we're only only interested in BERT's output for the [CLS] token, so we select that slice of the cube and discard everything else.



Slice the 3D tensor to get the 2D tensor of interest:

```
# Slice the output for the first position for all the sequences, take all hidden unit outputs
features = last_hidden_states[0][:,0,:].numpy()
```

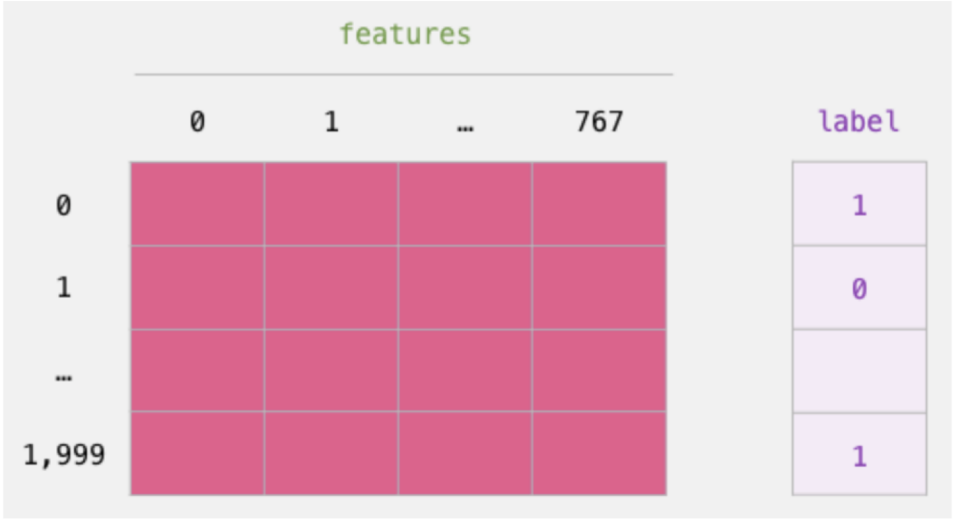
And now `features` is a 2d numpy array containing the sentence embeddings of all the sentences in our dataset.



The tensor we sliced from BERT's output

Dataset for Logistic Regression

The output of BERT can used to train the Logistic regression model. The 768 columns are the features and the 2,000 text labels are from the initial dataset.



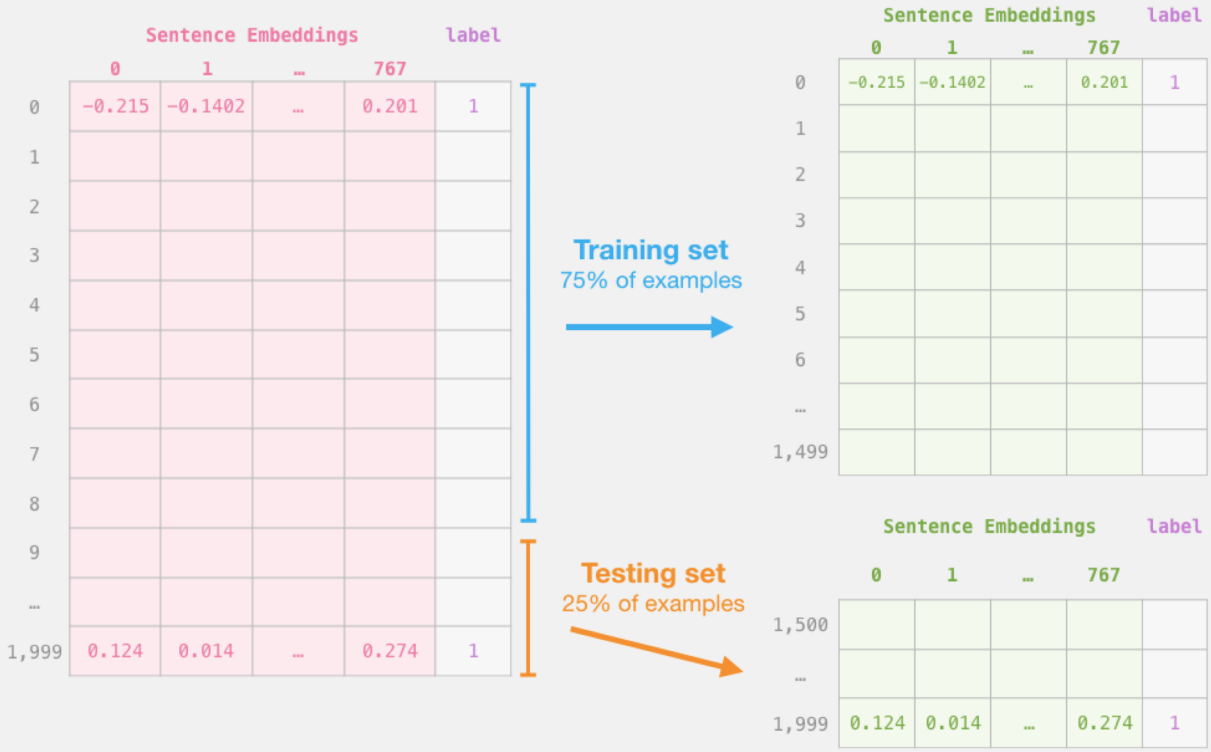
The labeled dataset we use to train the Logistic Regression. The features are the output vectors of BERT for the [CLS] token (position #0) that we sliced in the previous figure. Each row corresponds to a sentence in our dataset, each column corresponds to the output of a hidden unit from the feed-forward neural network at the top transformer block of the Bert/DistilBERT model.

After doing the traditional train/test split of machine learning, declare the Logistic Regression model and train it against the dataset.

```
labels = df[1]
train_features, test_features, train_labels, test_labels = train_test_split(features, labels)
```

This splits the dataset into training/testing sets:

Step #2: Test/Train Split for model #2, logistic regression



Next, we train the Logistic Regression model on the training set.

```
lr_clf = LogisticRegression()
lr_clf.fit(train_features, train_labels)
```

Now that the model is trained, we can score it against the test set:

```
lr_clf.score(test_features, test_labels)
```

Which shows the model achieves around 81% accuracy.

Score Benchmarks

- For reference, the highest accuracy score using the best ML models for this dataset is currently **96.8**.
- DistilBERT can be trained to improve its score on this task – a process called fine-tuning which updates BERT's weights to make it achieve a better performance in the sentence classification (which we can call the *downstream task*).
- The fine-tuned DistilBERT turns out to achieve an accuracy score of **90.7**.
- The full size BERT model achieves **94.9**.

The Notebook

Dive right into the notebook or run it on colab.

And that's it! That's a good first contact with BERT. The next step would be to head over to the documentation and try your hand at fine-tuning. You can also go back and switch from distilBERT to BERT and see how that works.

Acknowledgements: Jay Alammer thanks to Clément Delangue, Victor Sanh, and the Huggingface team for providing feedback to earlier versions of this tutorial.

Written on November 26, 2019

Acknowledgements & Attribution

Alammar, Jay (2018). The Illustrated Transformer [Blog post].

Retrieved from https://jalammar.github.io/illustrated-transformer/