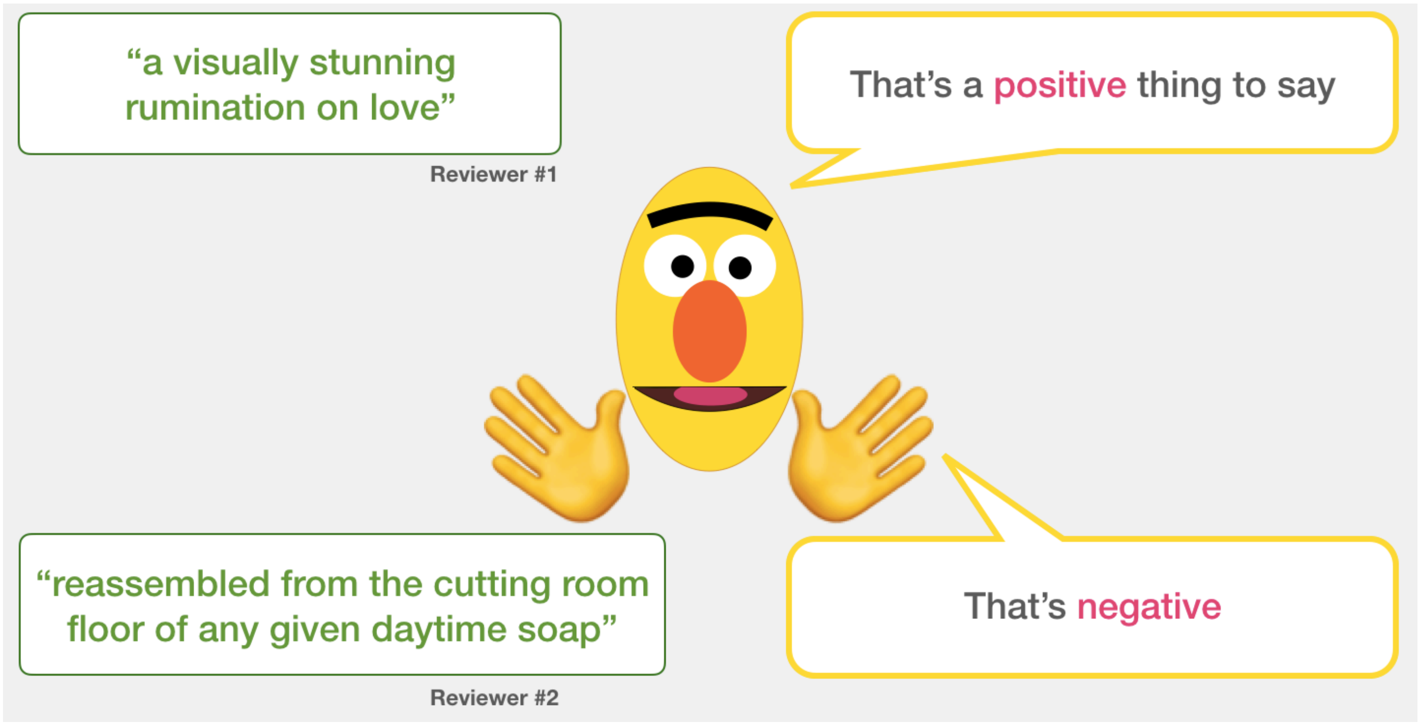
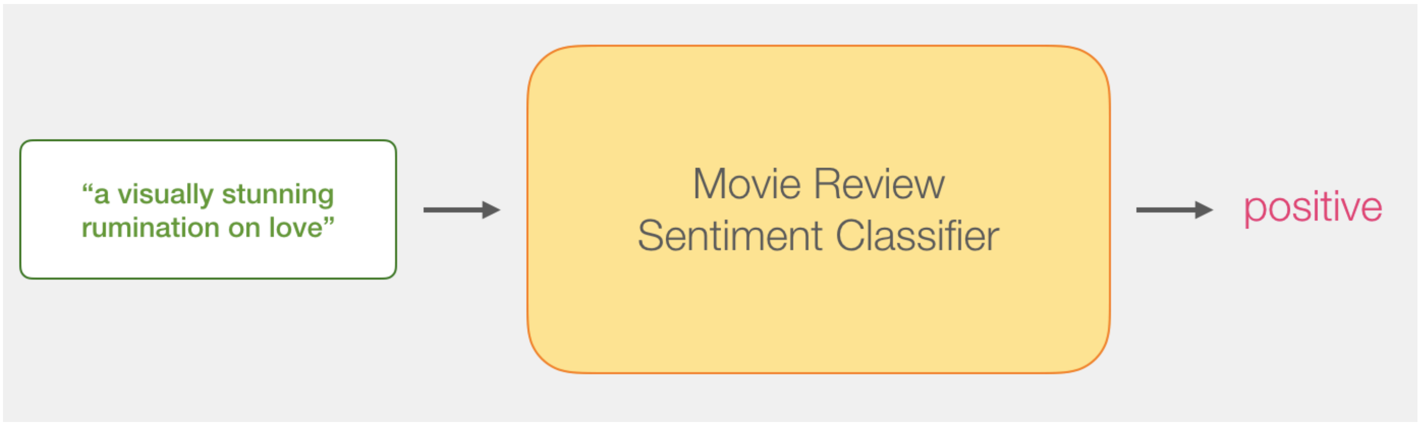
Getting Started with BERT


"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

In this notebook, we'll work with a pre-trained deep learning model to analyze text. We'll use the model's output to classify sentences, specifically determining whether each one conveys a positive or negative sentiment. The text we'll be analyzing comes from a list of film review sentences, and our task is to classify each sentence as either "positive" or "negative."

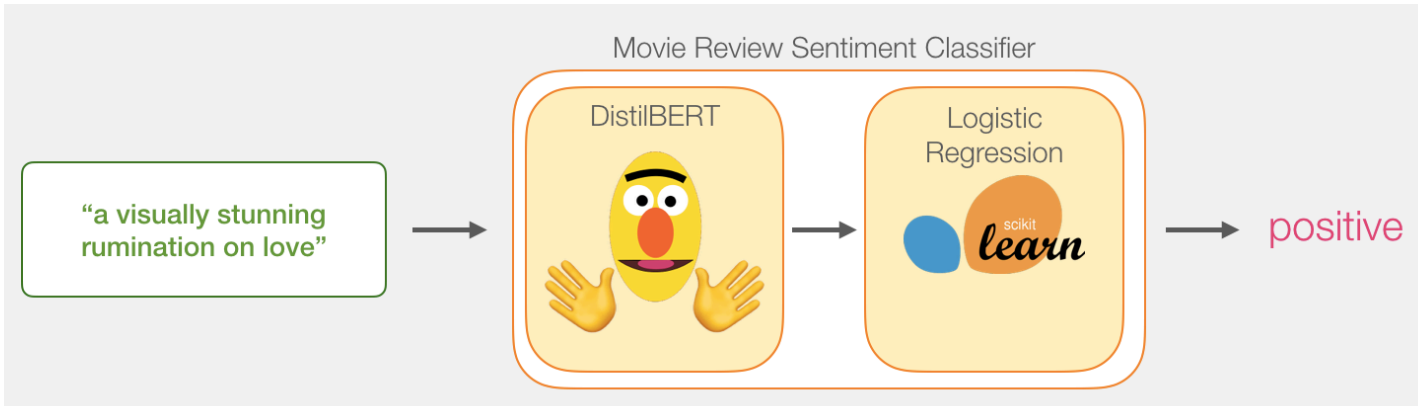
Models: Sentence Sentiment Classification

Our goal is to build a model that takes in a sentence from our dataset and classifies it as either 1 (positive sentiment) or 0 (negative sentiment). Here's a simple illustration of the process:


"a visually stunning rumination on love"
Movie Review Sentiment Classifier
positive

Behind the scenes, the model consists of two parts:

- DistilBERT: This is a smaller, faster version of the well-known BERT model, developed by HuggingFace. DistilBERT processes each sentence and extracts meaningful information, passing it along to the next step.
- Logistic Regression: The output from DistilBERT is then fed into a simple Logistic Regression model (from scikit-learn) to classify the sentence as positive (1) or negative (0). The information passed between the two models is a vector of size 768, which serves as a numerical representation (or embedding) of the sentence that we use for classification.


Movie Review Sentiment Classifier
"a visually stunning rumination on love"
DistilBERT
Logistic Regression
scikit learn
positive

Dataset

The dataset we will use in this example is SST2, which 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

Installing the transformers library

Let's start by installing the huggingface transformers library so we can load our deep learning NLP

```
!pip install transformers
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import torch
import transformers as ppb
import warnings
warnings.filterwarnings('ignore')
```

Importing the dataset

We'll use pandas to read the dataset and load it into a dataframe.

```
df = pd.read_csv('https://github.com/clairett/pytorch-sentiment-classification/raw/master/da
```

For performance reasons, we'll only use 2,000 sentences from the dataset

```
batch_1 = df[:2000]
```

We can ask pandas how many sentences are labeled as "positive" (value 1) and how many are labeled "negative" (having the value 0)

```
batch_1[1].value_counts()
```

Loading the Pre-trained BERT model

Let's now load a pre-trained BERT model.

```
# For DistilBERT:
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.DistilBertToker

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

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

Right now, the variable `model` holds a pretrained distilBERT model -- a version of BERT that is smaller, but much faster and requires a lot less RAM.

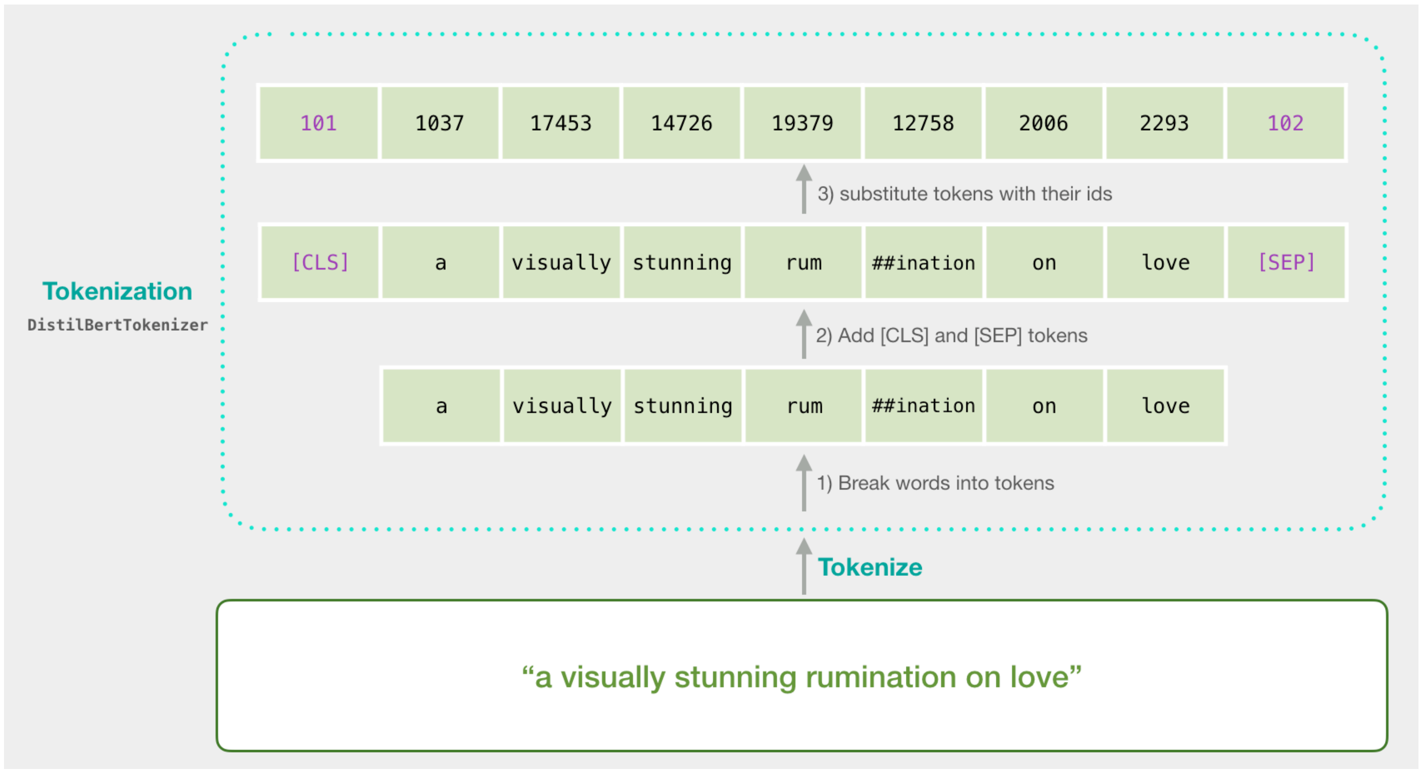
Model #1: Preparing the Dataset

Before we can pass our sentences to BERT, we need to do a bit of preprocessing to format them correctly.

Tokenization

The first step is tokenization—this means breaking the sentences into words and subwords in a way that BERT can work with.

```
tokenized = batch_1[0].apply((lambda x: tokenizer.encode(x, add_special_tokens=True)))
```



Padding

After tokenization, `tokenized` is a list of sentences -- each sentences is represented as a list of tokens. We want BERT to process our examples all at once (as one batch). It's just faster that way. For that reason, we need to pad all lists to the same size, so we can represent the input as one 2-d array, rather than a list of lists (of different lengths).

```
max_len = 0
for i in tokenized.values:
    if len(i) > max_len:
        max_len = len(i)

padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
```

Our dataset is now in the `padded` variable, we can view its dimensions below:

```
np.array(padded).shape
```

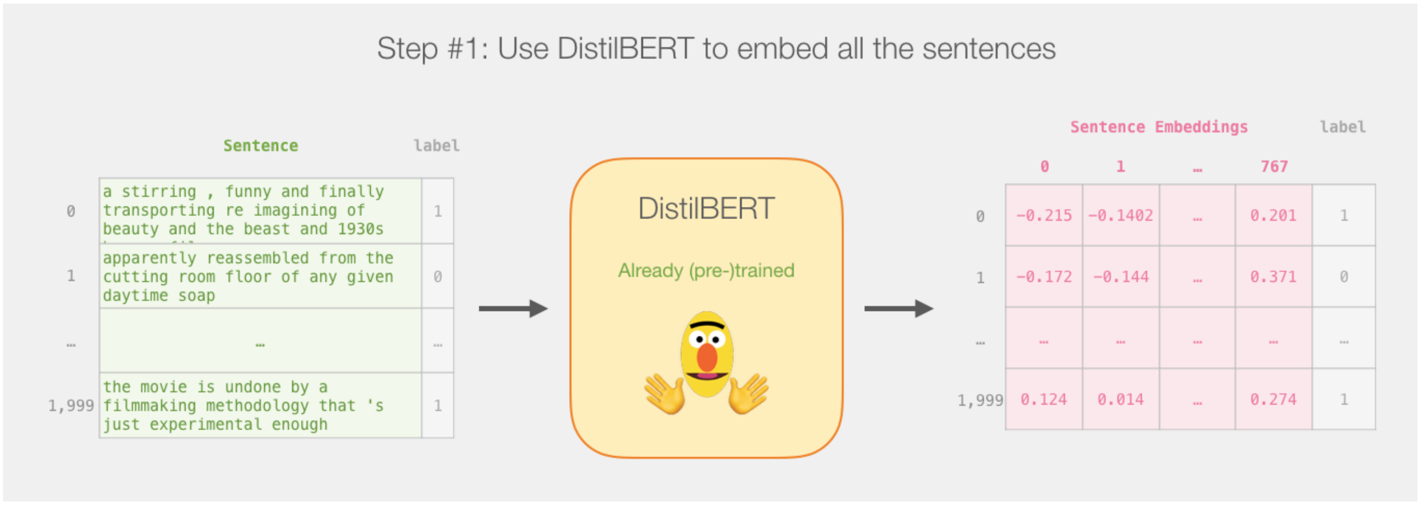
Masking

If we directly send `padded` to BERT, that would slightly confuse it. We need to create another variable to tell it to ignore (mask) the padding we have added when it's processing its input. That's what attention_mask is:

```
attention_mask = np.where(padded != 0, 1, 0)
attention_mask.shape
```

Model #1: And Now, Deep Learning!

Now that we have our model and inputs ready, let's run our model!



	Sentence	label
0	a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s	1
1	apparently reassembled from the cutting room floor of any given daytime soap	0
…	…	…
1,999	the movie is undone by a filmmaking methodology that 's just experimental enough	1

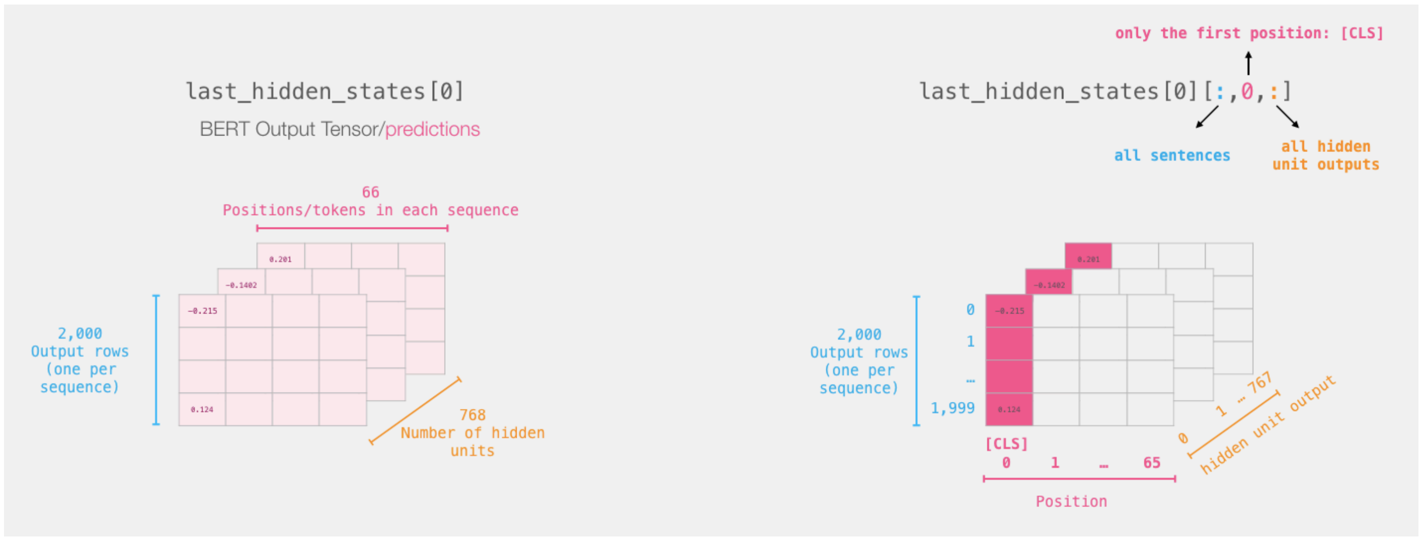
	Sentence Embeddings 0	1	…	767	label
0	-0.215	-0.1402	…	0.201	1
1	-0.172	-0.144	…	0.371	0
…	…	…	…	…	…
1,999	0.124	0.014	…	0.274	1

The `model()` function runs our sentences through BERT. The results of the processing will be returned into `last_hidden_states`.

```
input_ids = torch.tensor(padded)
attention_mask = torch.tensor(attention_mask)

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

Let's slice only the part of the output that we need. That is the output corresponding the first token of each sentence. The way BERT does sentence classification, is that it adds a token called `[CLS]` (for classification) at the beginning of every sentence. The output corresponding to that token can be thought of as an embedding for the entire sentence.



```
features = last_hidden_states[0][:,0,:].numpy()
```

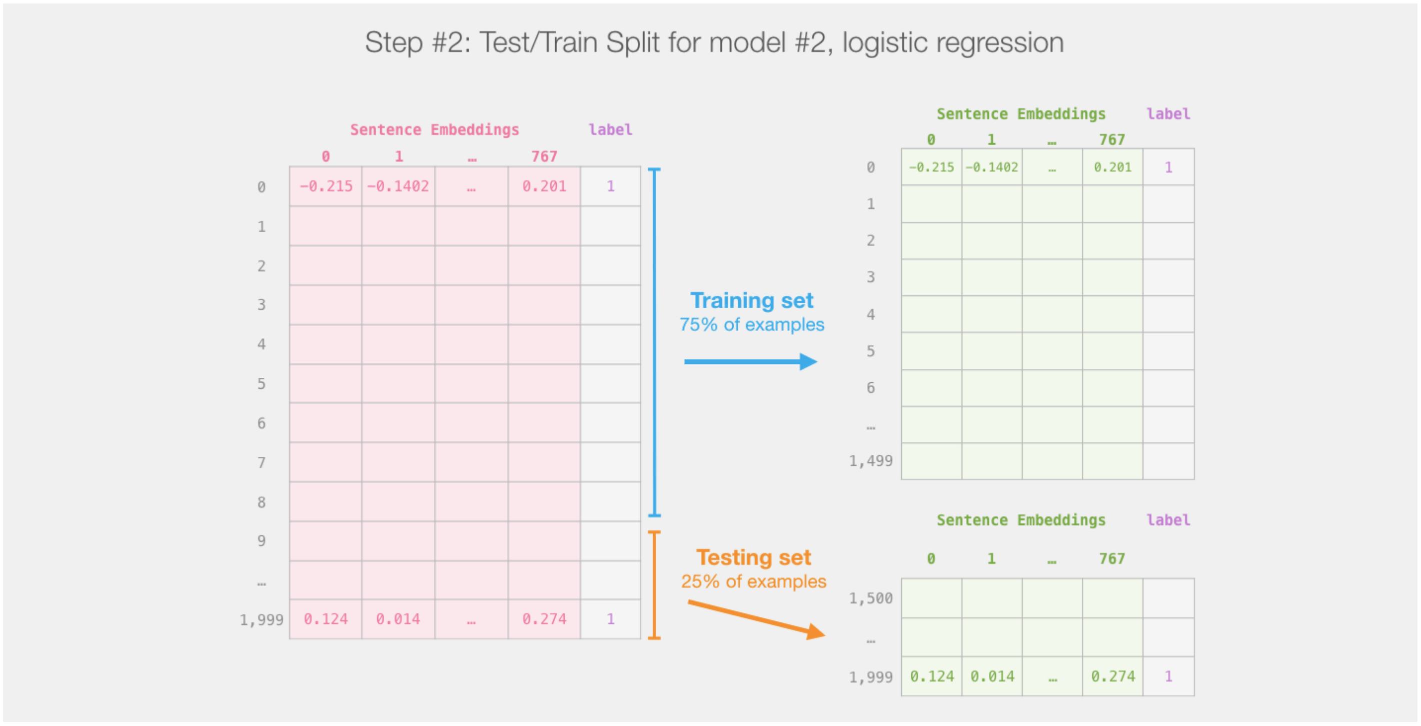
The labels indicating which sentence is positive and negative now go into the `labels` variable

```
labels = batch_1[1]
```

Model #2: Train/Test Split

Let's now split our datset into a training set and testing set (even though we're using 2,000 sentences from the SST2 training set).

```
train_features, test_features, train_labels, test_labels = train_test_split(features, labels
```



[Bonus] Grid Search for Parameters

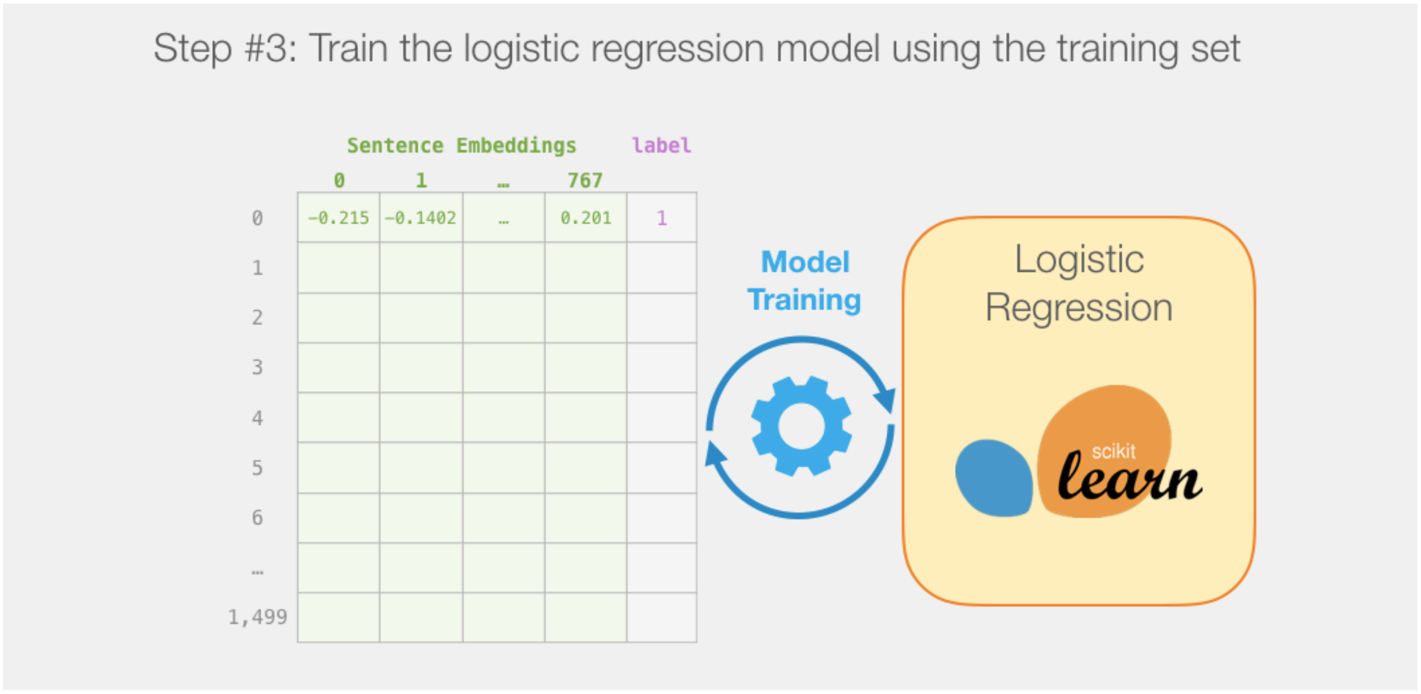
We can dive into Logistic regression directly with the Scikit Learn default parameters, but sometimes it's worth searching for the best value of the C parameter, which determines regularization strength.

```
# parameters = {'C': np.linspace(0.0001, 100, 20)}
# grid_search = GridSearchCV(LogisticRegression(), parameters)
# grid_search.fit(train_features, train_labels)

# print('best parameters: ', grid_search.best_params_)
# print('best scrores: ', grid_search.best_score_)
```

We can now train the LogisticRegression model. If you choose to do the gridsearch, you can plug the value of C into the model declaration (e.g. `LogisticRegression(C=5.2)`).

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



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

Evaluating Model #2

How good is this score? What can we compare it against? Let's first look at a dummy classifier:

So now well does our model do in classifying sentences? One way is to check the accuracy against

```
from sklearn.dummy import DummyClassifier
clf = DummyClassifier()

scores = cross_val_score(clf, train_features, train_labels)
print("Dummy classifier score: %0.3f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

So our model clearly does better than a dummy classifier. But how does it compare against the best models?

SST2 scores

For reference, the highest accuracy score 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 this sentence classification task (which we can call the downstream task).