# Final Project Explainable AI

Let's start by importing all the necesarry libraries. If you do not have all of these libraries downloaded, have a look at the ReadME file.

In [1]:
```
# For the classifier
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline
from accelerate import init_empty_weights
import numpy as np
from sklearn.metrics import accuracy_score
```

```
C:\Users\maasl\anaconda3\envs\XAI\lib\site-packages\transformers\utils\generic.py:441: F
utureWarning: `torch.utils._pytree._register_pytree_node` is deprecated. Please use `tor
ch.utils._pytree.register_pytree_node` instead.
  _torch_pytree._register_pytree_node(
C:\Users\maasl\anaconda3\envs\XAI\lib\site-packages\transformers\utils\generic.py:309: F
utureWarning: `torch.utils._pytree._register_pytree_node` is deprecated. Please use `tor
ch.utils._pytree.register_pytree_node` instead.
  _torch_pytree._register_pytree_node(
C:\Users\maasl\anaconda3\envs\XAI\lib\site-packages\transformers\utils\generic.py:309: F
utureWarning: `torch.utils._pytree._register_pytree_node` is deprecated. Please use `tor
ch.utils._pytree.register_pytree_node` instead.
  _torch_pytree._register_pytree_node(
```

In [2]:
```
# For LIME
from lime.lime_text import LimeTextExplainer
```

In [3]:
```
# For the attention-based method
import torch
import matplotlib.pyplot as plt
import seaborn as sns
```

## Dataset

I am using the IMDB dataset, which consists of movie reviews. It contains 50000 records, divided into 25000 for training and 25000 for testing. The records have an according label, which indicates of the review is positive or negative. The dataset can be found here: https://www.kaggle.com/code/trentpark/data-analysis-basics-imdb-dataset

In [4]:
```
# Load IMDB dataset
dataset = load_dataset("imdb")
```

In [5]:
```
# Extract the train and test data (only test data used)
train_data = dataset["train"]
test_data = dataset["test"]
```

The classification model used is a DistilBERT model pretrained on the IMDB dataset. For more information see https://huggingface.co/textattack/distilbert-base-uncased-imdb.

In [6]:
```
model_name = "textattack/distilbert-base-uncased-imdb"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)
```

```
C:\Users\maasl\anaconda3\envs\XAI\lib\site-packages\huggingface_hub\file_download.py:89
```

```
6: FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0.
Downloads always resume when possible. If you want to force a new download, use `force_d
ownload=True`.
  warnings.warn(
```

The Bert model doesnt take inputs longer than 512 characters, so we will truncate reviews which are longer than 512 words.

In [7]:
```
# Creating the text classification pipeline
classifier = pipeline(
    "text-classification",
    model=model,
    tokenizer=tokenizer,
    truncation=True,         #truncate long reviews to 512 tokens
    max_length=512,
    return_all_scores=False
)
```

```
C:\Users\maasl\anaconda3\envs\XAI\lib\site-packages\transformers\pipelines\text_classifi
cation.py:105: UserWarning: `return_all_scores` is now deprecated,  if want a similar fu
nctionality use `top_k=None` instead of `return_all_scores=True` or `top_k=1` instead of
`return_all_scores=False`.
  warnings.warn(
```

We want to test whether the model functions properly. To get a rough estimate on its accuracy on the dataset, we randomly sample (with seed for reproducability) a subset of 300 from the dataset, and compute their accuracy.

In [8]:
```
# Randomly sample from test data with fixed seed for reproducability
np.random.seed(42)

# We are using a sample of 300 to get an estimate of the model's accuracy
indices = np.random.choice(len(dataset["test"]), size=300, replace = False)
subset = dataset["test"].select(indices)

# Extracting texts from the subset, for later use
texts = [example['text'] for example in subset]
true_labels = [example['label'] for example in subset]  # 0 = neg, 1 = pos
```

In [9]:
```
# Predict using the classifier
predictions = classifier(texts)
```

In [10]:
```
# Example input texts passed through the classifier. LABEL_0 is negative and LABEL_1 pos
print(classifier("This movie was absolutely terrible. I hated it"))
print(classifier("This was one of the best films I have ever seen"))
```

```
[{'label': 'LABEL_0', 'score': 0.998586893081665}]
[{'label': 'LABEL_1', 'score': 0.9978604912757874}]
```

Estimating the accuracy

In [11]:
```
# Convert model predictions POSITIVE/NEGATIVE to 1/0
predicted_labels = [1 if p["label"] == "LABEL_1" else 0 for p in predictions]

# Estimate accuracy
accuracy = accuracy_score(true_labels, predicted_labels)
print(f"Accuracy on 300 random IMDB test samples: {accuracy:.2f}")
```

```
Accuracy on 300 random IMDB test samples: 0.91
```

Although only on 300 samples, we can see that with an accuracy of 0.91 the model definitely performs better than chance, so everything is working properly

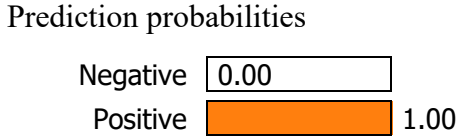
# LIME explainer

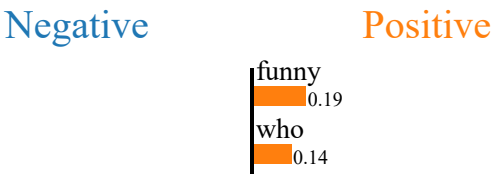
The Hugging Face pipeline only returns the predicted label and its probability, while lime wants class probabilities for all classes for each instance. Furthermore, LIME needs a function that can process all the perturbated inputs. We thus have to create a wrapper function.

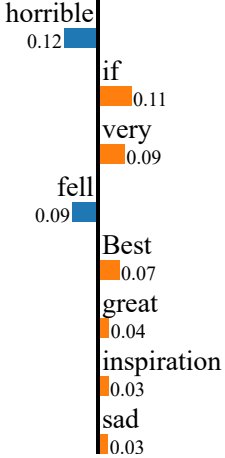
In [12]:
```python
def predict_proba(text):
    '''
    This functions acts as a wrapper function for the LIME classifier

    Parameters:
        - text (list of str): A list of input strings which are
        perturbations of the original input

    Returns:
        - np.ndarray: NumPy array of shape (n_samples, 2), where rows contain
        the probability for the negative and positive labels respectively
    '''
    outputs = classifier(text)
    return np.array([
        [1 - pred['score'], pred['score']] if pred['label'] == "LABEL_1"
        else [pred['score'], 1 - pred['score']]
        for pred in outputs
    ])
```

In [13]:
```python
# Initiating explainer with display-purpose class names
explainer = LimeTextExplainer(class_names = ["Negative", "Positive"])
```

In [14]:
```python
example_1 = texts[10]
# print(example_1)

explanation = explainer.explain_instance(
    text_instance=example_1,
    classifier_fn=predict_proba,
    num_features=10,
    top_labels=1,
    num_samples = 5000)

explanation.show_in_notebook(text=True)
```






## Text with highlighted words

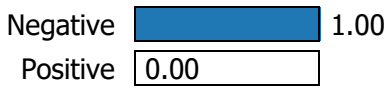
This was very funny, even if it fell apart a little at the end. Does not go overboard with homage after to Hitchcock - Owen (Danny DeVito) was lucky he had "Strangers on A Train" playing at the local cinema, so the movie flat out tells you that that was the inspiration. |br /||br /|DeVito is very funny but also a little sad. He has no friends and all he wants to do is write and have someone like his writing. His teacher, Billy Crystal, is going through some serious writers block of his own and his wife has stolen his book and made it her own success, which also has him frustrated a great deal.|br /||br /|Best parts are the book proposal by Mr. Pinsky ("One Hundred Girls I'd Like to Pork") and all scenes with Anne Ramsey, who is so horrible that even Mother Theresa would have wanted to kill her, too!

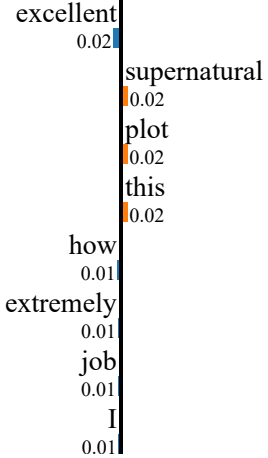
In the example above, we can see a positive review. The 10 most influential words that LIME finds are shown. We can see that words like 'funny', 'very', 'best' contribute positively to the prediction (that it's positive). Words like 'horrible' and 'fell' contribute negatively.

Negative example

In [15]:
```
example_2 = texts[5]

explanation2 = explainer.explain_instance(
    text_instance=example_2,
    classifier_fn=predict_proba,
    num_features=10,
    top_labels=1,
    num_samples = 5000)

explanation2.show_in_notebook(text=True)
```

Prediction probabilities




## Text with highlighted words

I thought this was an extremely bad movie. The whole time I was watching this movie I couldn't help but think over and over how bad it is, and how that was $3.69 down the drain. The plot was so jumpy. They did an excellent job at the beginning of explaining who dated who in high school, but they never really explained anything after that. Was it a supernatural thriller? Was it a regular thriller? Apparently you can decide for yourself, because they didn't see the need to explain. I understood basically what happened, I think. What I got confused about was all of it prior, what was the deal with the bloody noses, phone calls, etc.? Was this guy coming back? Was the wife channeling "Carrie" or something? Who knows? You certainly won't after watching this movie.

In the example above we can see a negative review. LIME mainly highlights words like bad and jumpy.

# Attention-based methods

BERT models use multi-head self-attention. Each attention head produces a matrix, which shows how much each token pays attention to other tokens. Inspecting these outputs, we can derive which tokens the model focussed the most on.

In [16]:
```
# Create a new model that outputs the attention weights
model_attention = AutoModelForSequenceClassification.from_pretrained(
    model_name, output_attentions = True
)
# The tokenizer stays the same

# Put model in eval mode
model_attention.eval()
```

```
C:\Users\maasl\anaconda3\envs\XAI\lib\site-packages\huggingface_hub\file_download.py:896: FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
```

Out[16]:
```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
```

```
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

Computing the attention weights for the first example.

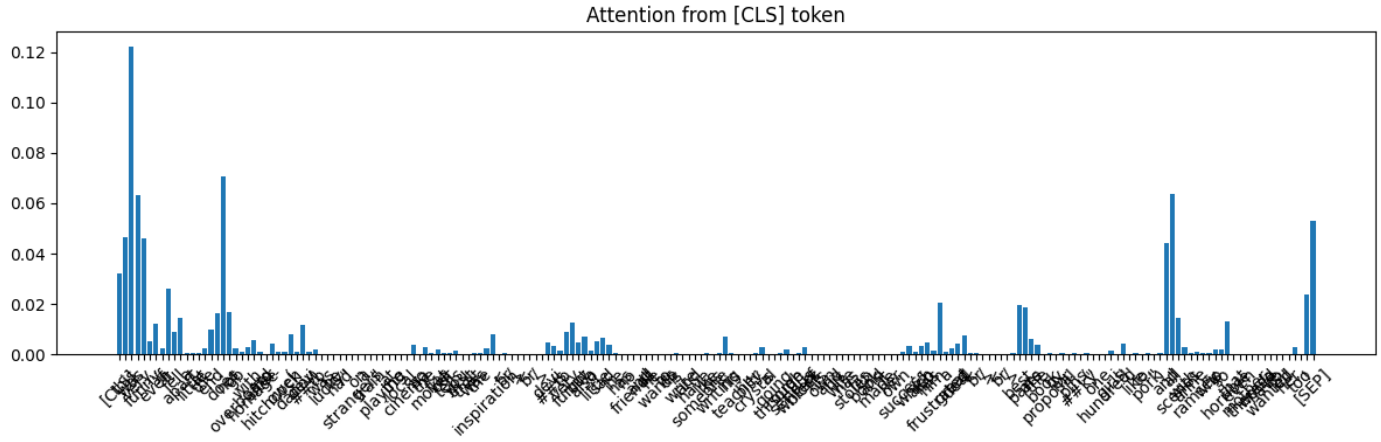
In [17]:
```
inputs1 = tokenizer(example_1, return_tensors = "pt", truncation = True)

with torch.no_grad():
    outputs = model_attention(**inputs1)  # extracting the inputs and passing them the t
    attentions = outputs.attentions
```

In [18]:
```
print(type(attentions))
print(len(attentions))
print(attentions[5].shape)
# print(inputs1)
```

```
<class 'tuple'>
6
torch.Size([1, 12, 196, 196])
```

We get a tuple with a length of 6, which means the DistilBERT model has 6 layers. To inspect the attention, we will use the last layer's weights, because this gives us the explanation of the decisions the model made. We will look at the attention of the [CLS] token, which is a special token added to the beginning of input text by BERT models. It acts as a summary.

In [19]:
```
# Grabbing the last attention tuple
last_att = attentions[-1][0]  # --> shape (12, 196, 196)

# Averaging the attention over all 12 heads.
avg_att = last_att.mean(dim = 0)  # --> shape (196,196)

cls_attn = avg_att[0]                    # --> shape (196,)
tokens = tokenizer.convert_ids_to_tokens(inputs1['input_ids'][0])

# Plot
plt.figure(figsize=(12, 4))
plt.bar(range(len(tokens)), cls_attn)
plt.xticks(range(len(tokens)), tokens, rotation=45)
plt.title("Attention from [CLS] token")
plt.tight_layout()
plt.show()
```



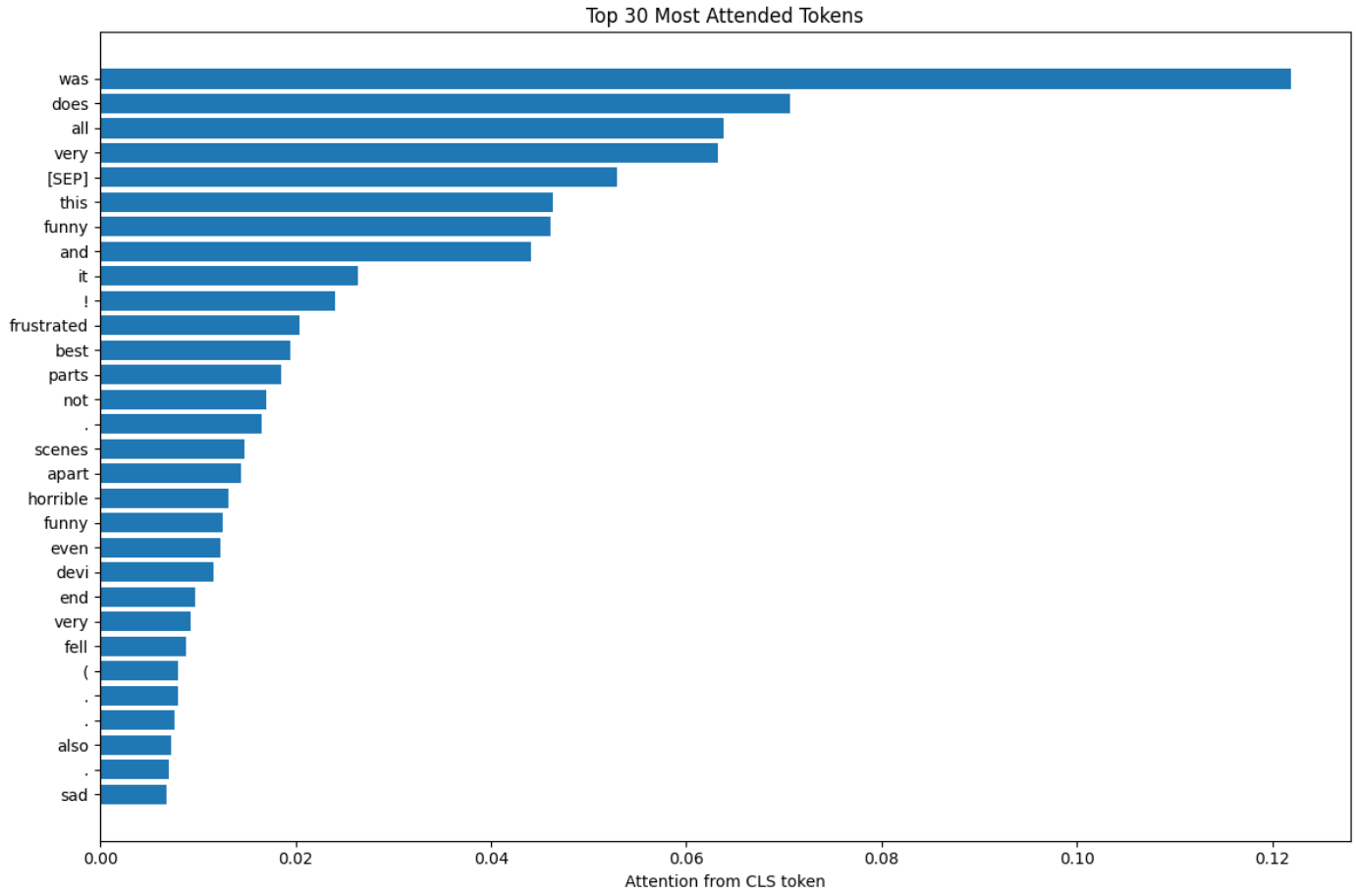

As we can see, there is too many words, so let's create a function where we can control the amount of tokens plotted.

In [20]:
```python
def plot_topk_cls(text, model, tokenizer, k=20):
    '''
    Visualises top-k tokens that the [CLS] token attends to in the final attention layer
    transformer model. It generates a horizontal bar plot showing the scores.
    '''
    # Tokenize the input
    inputs = tokenizer(text, return_tensors="pt", truncation=True)

    # Forward pass with attention output
    with torch.no_grad():
        outputs = model(**inputs, output_attentions=True)

    last_attn = outputs.attentions[-1][0]    # Get attentions from last layer
    avg_attn = last_attn.mean(dim=0)        # Average over all 12 heads
    cls_attn = avg_attn[0]                  # attention from [CLS] token

    # Get tokens
    tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])

    # Get top-k token indices excluding CLS
    topk_id = torch.topk(cls_attn[1:], k=k).indices + 1
    topk_scores = cls_attn[topk_id]
    topk_tokens = [tokens[i] for i in topk_id]

    # Plot
    plt.figure(figsize=(12, 8))
    plt.barh(range(k), topk_scores.tolist())
    plt.yticks(range(k), topk_tokens)
    plt.gca().invert_yaxis()
    plt.xlabel("Attention from CLS token")
    plt.title(f"Top {k} Most Attended Tokens")
    plt.tight_layout()
    plt.show()
```

In [21]:
```python
print(example_1)
plot_topk_cls(example_1, model_attention, tokenizer, k = 30)
```

```
This was very funny, even if it fell apart a little at the end. Does not go overboard wi
th homage after to Hitchcock - Owen (Danny DeVito) was lucky he had "Strangers on A Trai
n" playing at the local cinema, so the movie flat out tells you that that was the inspir
ation. <br /><br />DeVito is very funny but also a little sad. He has no friends and all
he wants to do is write and have someone like his writing. His teacher, Billy Crystal, i
s going through some serious writers block of his own and his wife has stolen his book a
nd made it her own success, which also has him frustrated a great deal.<br /><br />Best
parts are the book proposal by Mr. Pinsky ("One Hundred Girls I'd Like to Pork") and all
```

```
scenes with Anne Ramsey, who is so horrible that even Mother Theresa would have wanted t
o kill her, too!
```




## Making an attention heatmap

Creating a function to extract again the top-k tokens based on [CLS], include only those tokens in the attention matrix, then showing the k by k heatmap, which shows how these tokens pay attention to eachother.

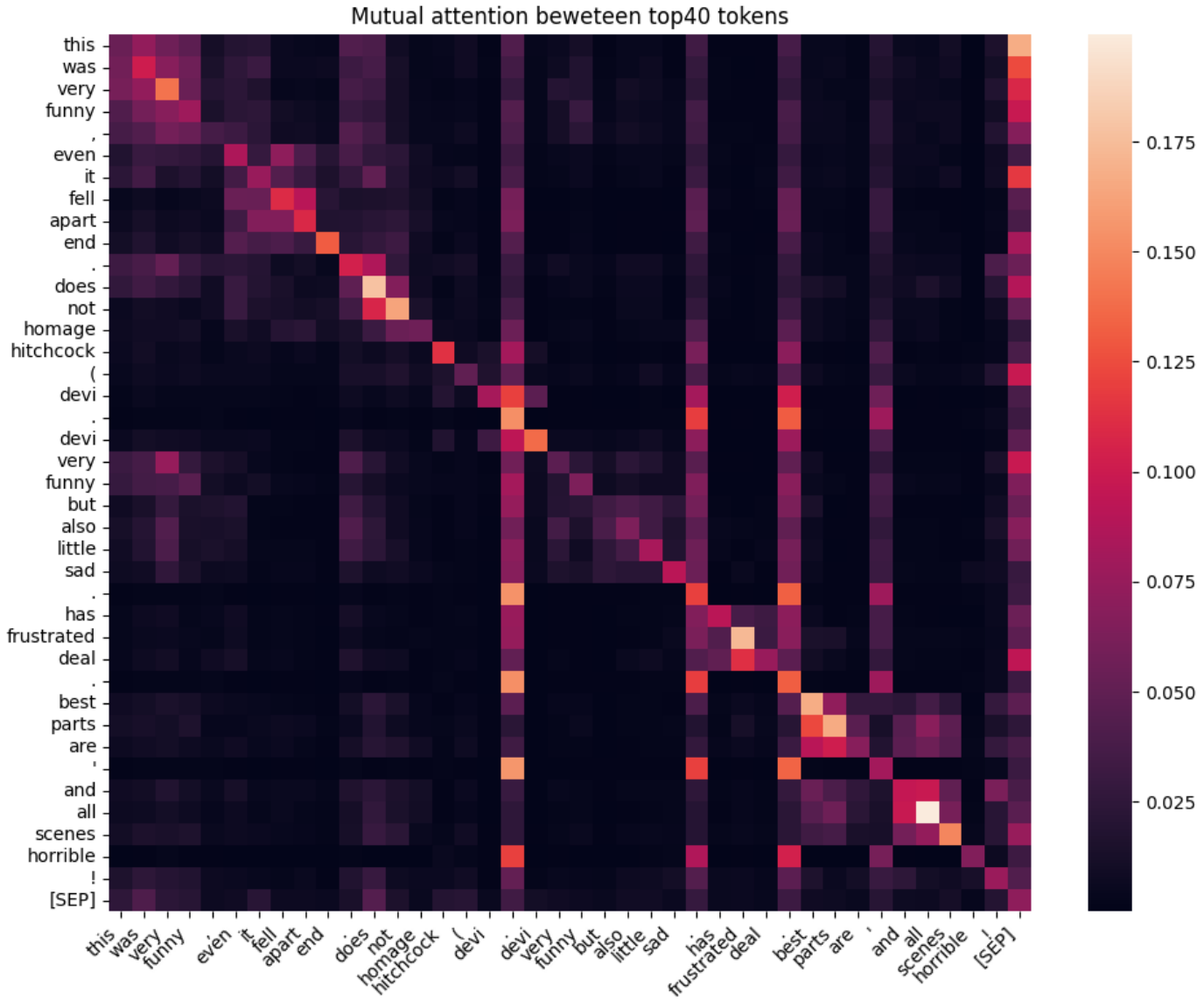
In [22]:

```python
def plot_cls_heatmap(text, model, tokenizer, k = 30):
    '''
    This function visualises the mutual attention scores between the top-k attended toke
    by the [CLS] token in the final attention layer of a transformer model. It displays
    as a k-by-k heatmap, where the rows are the attending tokens and columns the attende
    '''
    # Tokenize the input
    inputs = tokenizer(text, return_tensors="pt", truncation=True)

    # Forward pass with attention output
    with torch.no_grad():
        outputs = model(**inputs, output_attentions=True)

    last_attn = outputs.attentions[-1][0]    # Get attentions from last layer
    avg_attn = last_attn.mean(dim=0)        # Average over all 12 heads
    cls_attn = avg_attn[0]                  # attention from [CLS] token

    # Get tokens
    tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])

    # excluding [CLS]
    topk_ids = torch.topk(cls_attn[1:], k=min(k, len(cls_attn) - 1)).indices + 1

    # Sorting so it's easier to only extract the top k tokens
    topk_ids = topk_ids.sort().values
```

```
    topk_tokens = [tokens[i] for i in topk_ids]
    matrix = avg_attn[topk_ids][:, topk_ids]

    # Plotting heatmap
    # Row: Attending token
    # Column: Attended-to token
    plt.figure(figsize = (10,8))
    sns.heatmap(matrix.numpy(), xticklabels = topk_tokens, yticklabels = topk_tokens)
    plt.xticks(rotation=45, ha='right')
    plt.title(f"Mutual attention beweteen top{k} tokens")
    plt.tight_layout()
    plt.show()
```

In [23]:
```
print(example_1)
plot_cls_heatmap(example_1, model_attention, tokenizer, k = 40)
```

```
This was very funny, even if it fell apart a little at the end. Does not go overboard wi
th homage after to Hitchcock - Owen (Danny DeVito) was lucky he had "Strangers on A Trai
n" playing at the local cinema, so the movie flat out tells you that that was the inspir
ation. <br /><br />DeVito is very funny but also a little sad. He has no friends and all
he wants to do is write and have someone like his writing. His teacher, Billy Crystal, i
s going through some serious writers block of his own and his wife has stolen his book a
nd made it her own success, which also has him frustrated a great deal.<br /><br />Best
parts are the book proposal by Mr. Pinsky ("One Hundred Girls I'd Like to Pork") and all
scenes with Anne Ramsey, who is so horrible that even Mother Theresa would have wanted t
o kill her, too!
```



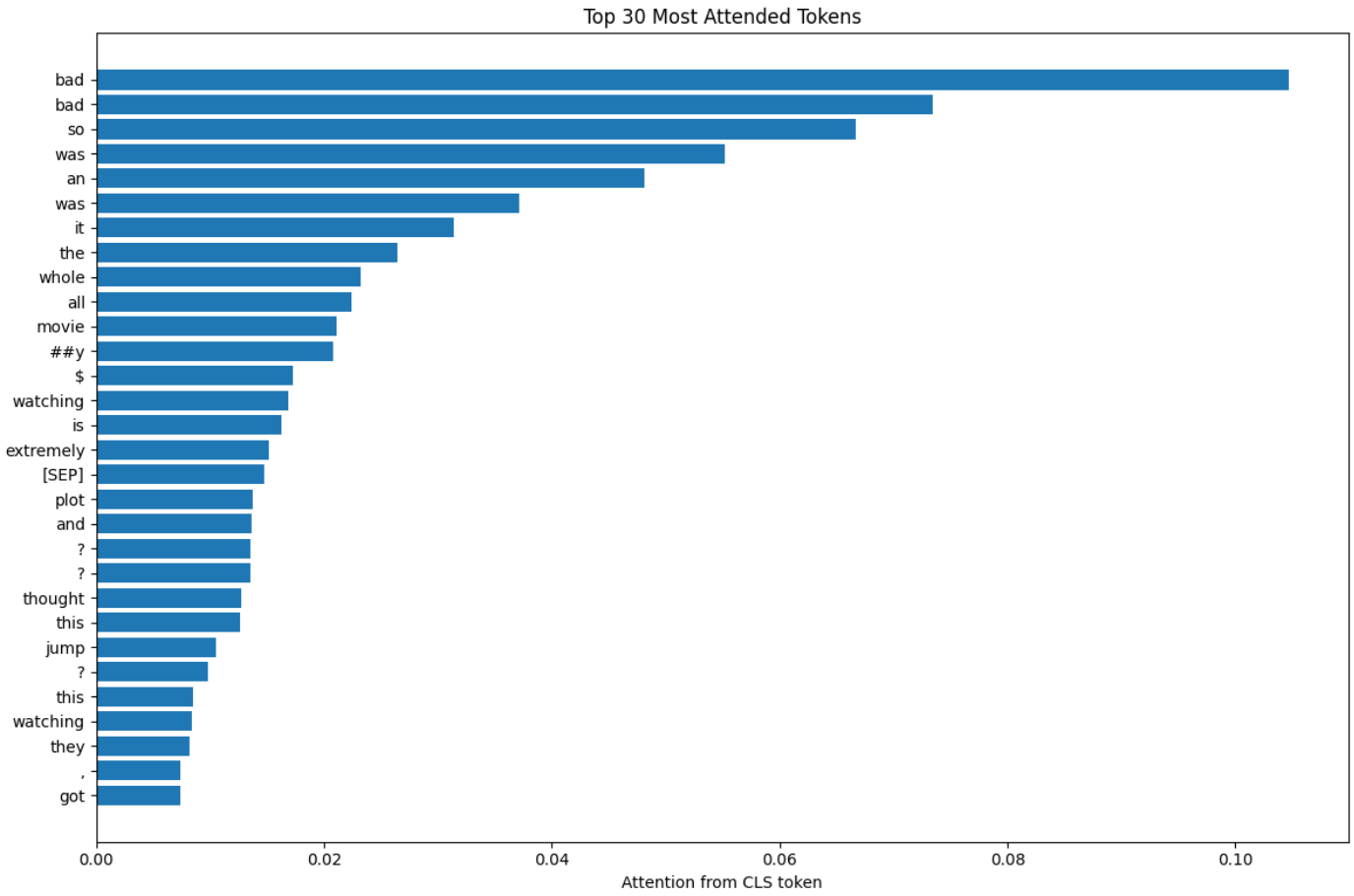

Above can be seen how words that are grouped together and are more related, tend to 'pay' more attention

to eachother in the model.

Now let's do the same for the second example

In [24]:
```
print(example_2)
plot_topk_cls(example_2, model_attention, tokenizer, k = 30)
```

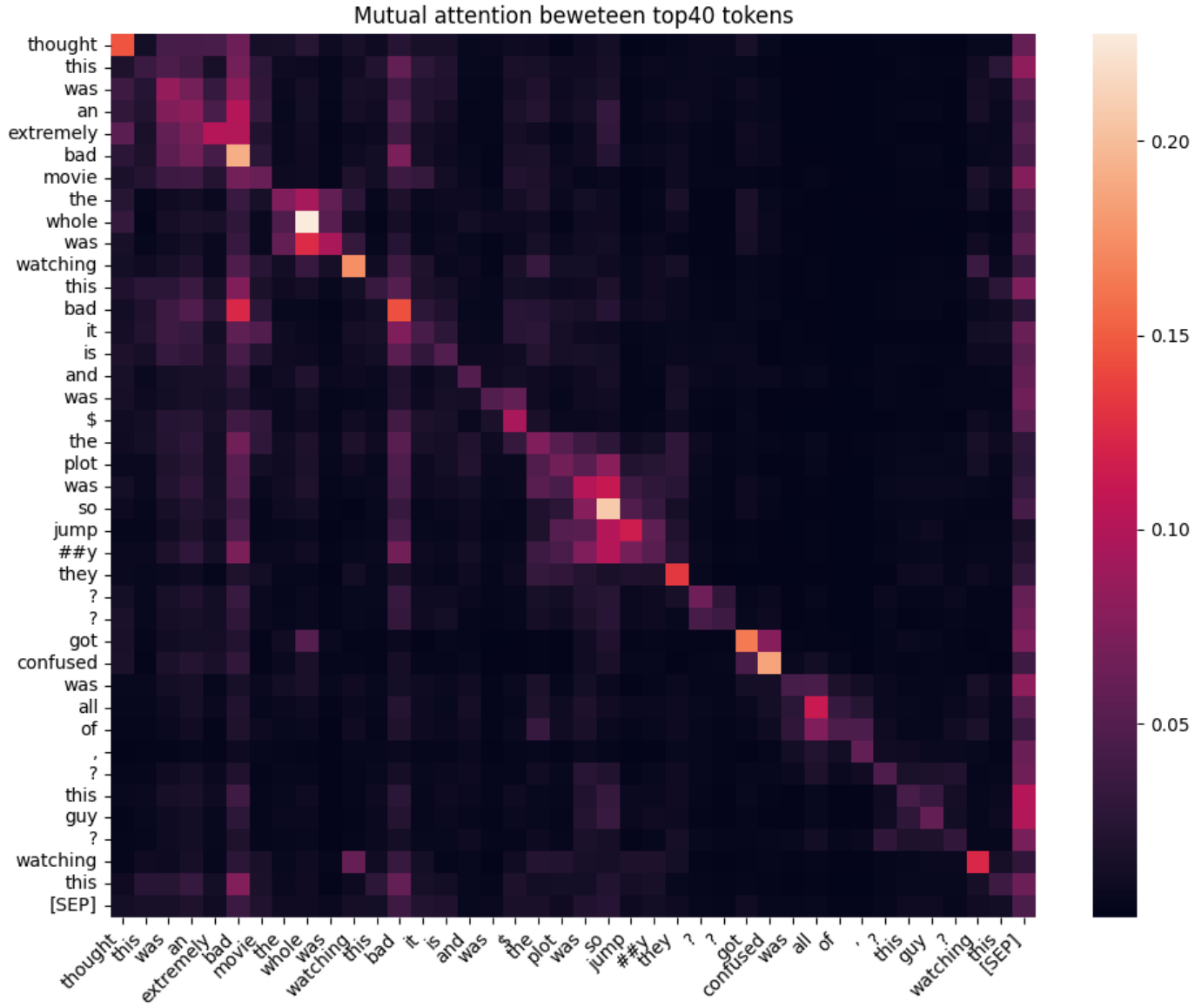
```
I thought this was an extremely bad movie. The whole time I was watching this movie I co
uldn't help but think over and over how bad it is, and how that was $3.69 down the drai
n. The plot was so jumpy. They did an excellent job at the beginning of explaining who d
ated who in high school, but they never really explained anything after that. Was it a s
upernatural thriller? Was it a regular thriller? Apparently you can decide for yourself,
because they didn't see the need to explain. I understood basically what happened, I thi
nk. What I got confused about was all of it prior, what was the deal with the bloody nos
es, phone calls, etc.? Was this guy coming back? Was the wife channeling "Carrie" or som
ething? Who knows? You certainly won't after watching this movie.
```




Since this is a negative review, most attention goes to the words 'bad', which makes a lot of sense.

In [25]:
```
print(example_2)
plot_cls_heatmap(example_2, model_attention, tokenizer, k=40)
```

```
I thought this was an extremely bad movie. The whole time I was watching this movie I co
uldn't help but think over and over how bad it is, and how that was $3.69 down the drai
n. The plot was so jumpy. They did an excellent job at the beginning of explaining who d
ated who in high school, but they never really explained anything after that. Was it a s
upernatural thriller? Was it a regular thriller? Apparently you can decide for yourself,
because they didn't see the need to explain. I understood basically what happened, I thi
nk. What I got confused about was all of it prior, what was the deal with the bloody nos
es, phone calls, etc.? Was this guy coming back? Was the wife channeling "Carrie" or som
ething? Who knows? You certainly won't after watching this movie.
```




Again, related words pay more attention to eachother.

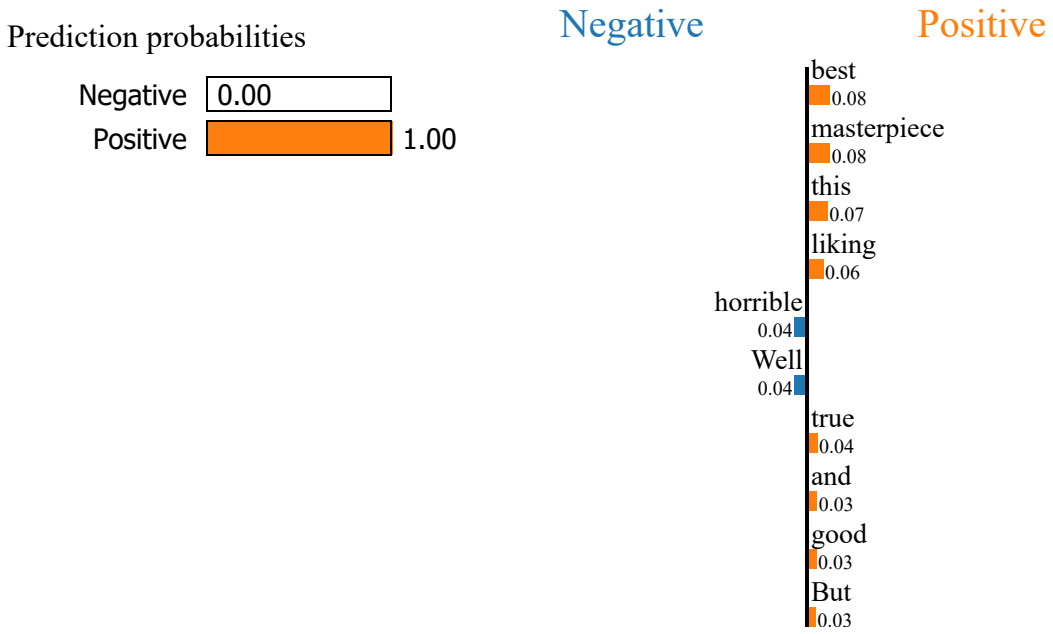
# Extra Examples

In [26]:
```
print(texts[30])
print("\n")
print(texts[250])
```

```
Now, the sci-fi channel original company has made some pretty crappy films (House of the
dead 2, All souls day, etc.) but when you leave the job entirely to horror master actor/
writer and now director, Bruce Campbell, you get one of the best damn made for TV indepe
ndent horror films ever made! I normally hate these movies, in my previous review, House
of the dead 2, I could not believe how horrible the film was! But somehow I took a likin
g for this film, a very good liking for this film. The violence is good and so is the bl
ack comedy in the film and I recommend you get it, a true Bruce Campbell masterpiece! We
ll, since there is only a few more lines left I can say whatever I want about this movi
e: IJAJKASIF JHJDJ NXD FNEHSD FHNCFNFVHS DJKEALJWSNS.UHHD SISHSNHF AHCNAKDJH HNDCHJNDNH
JACND HCHJNNHW JHJ NASHDNFHCKA FHNKHAD SAKASDADJ FJKDFA


i have to admit thanks to this movie i'm now afraid of mannequins. hahaha.<br /><br />bu
t yes, first off the acting in this movie at least by my standard is pretty swell. most
of the actors are pretty decent in their roles. the script also seems to be pretty good
too, sure some cheesy stuff in there but also some decently written character and some d
amn scary scenes. i STILL get shivers thinking about that one scene with the dude and th
```

e mannequins. brrr.<br /><br />yeah, I'll say you should check this movie out it's prett
y good, and very entertaining. a good watch. 8/10

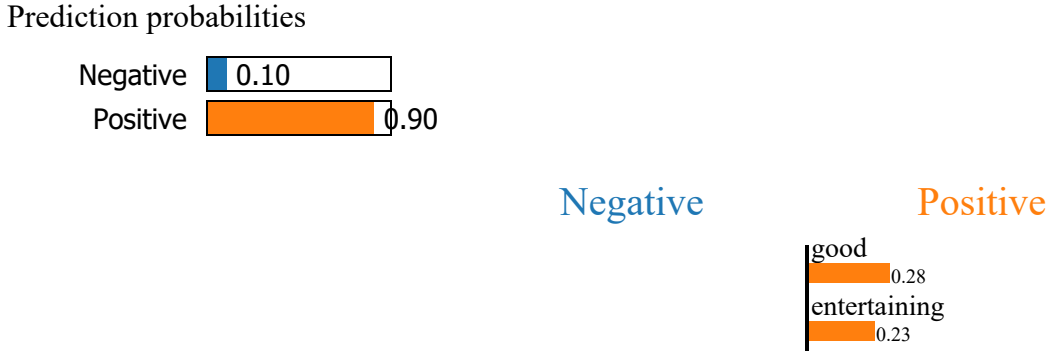
In [27]:
```
extra_explanation1 = explainer.explain_instance(
    text_instance=texts[30],
    classifier_fn=predict_proba,
    num_features=10,
    top_labels=1,
    num_samples = 2500)

extra_explanation1.show_in_notebook(text=True)
```

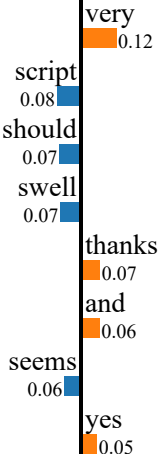



## Text with highlighted words

Now, the sci-fi channel original company has made some pretty crappy films (House of the dead 2, All souls day, etc.) but when you leave the job entirely to horror master actor/writer and now director, Bruce Campbell, you get one of the best damn made for TV independent horror films ever made! I normally hate these movies, in my previous review, House of the dead 2, I could not believe how horrible the film was! But somehow I took a liking for this film, a very good liking for this film. The violence is good and so is the black comedy in the film and I recommend you get it, a true Bruce Campbell masterpiece! Well, since there is only a few more lines left I can say whatever I want about this movie: IJAJKASIF JHJDJ NXD FNEHSD FHNCFNFVHS DJKEALJWSNS.UHHD SISHSNHF AHCNAKDJH HNDCHJNDNH JACND HCHJNNHW JHJ NASHDNFHCKA FHNKHAD SAKASDADJ FJKDFA

In [28]:
```
extra_explanation2 = explainer.explain_instance(
    text_instance=texts[250],
    classifier_fn=predict_proba,
    num_features=10,
    top_labels=1,
    num_samples = 2500)

extra_explanation2.show_in_notebook(text=True)
```






# Text with highlighted words

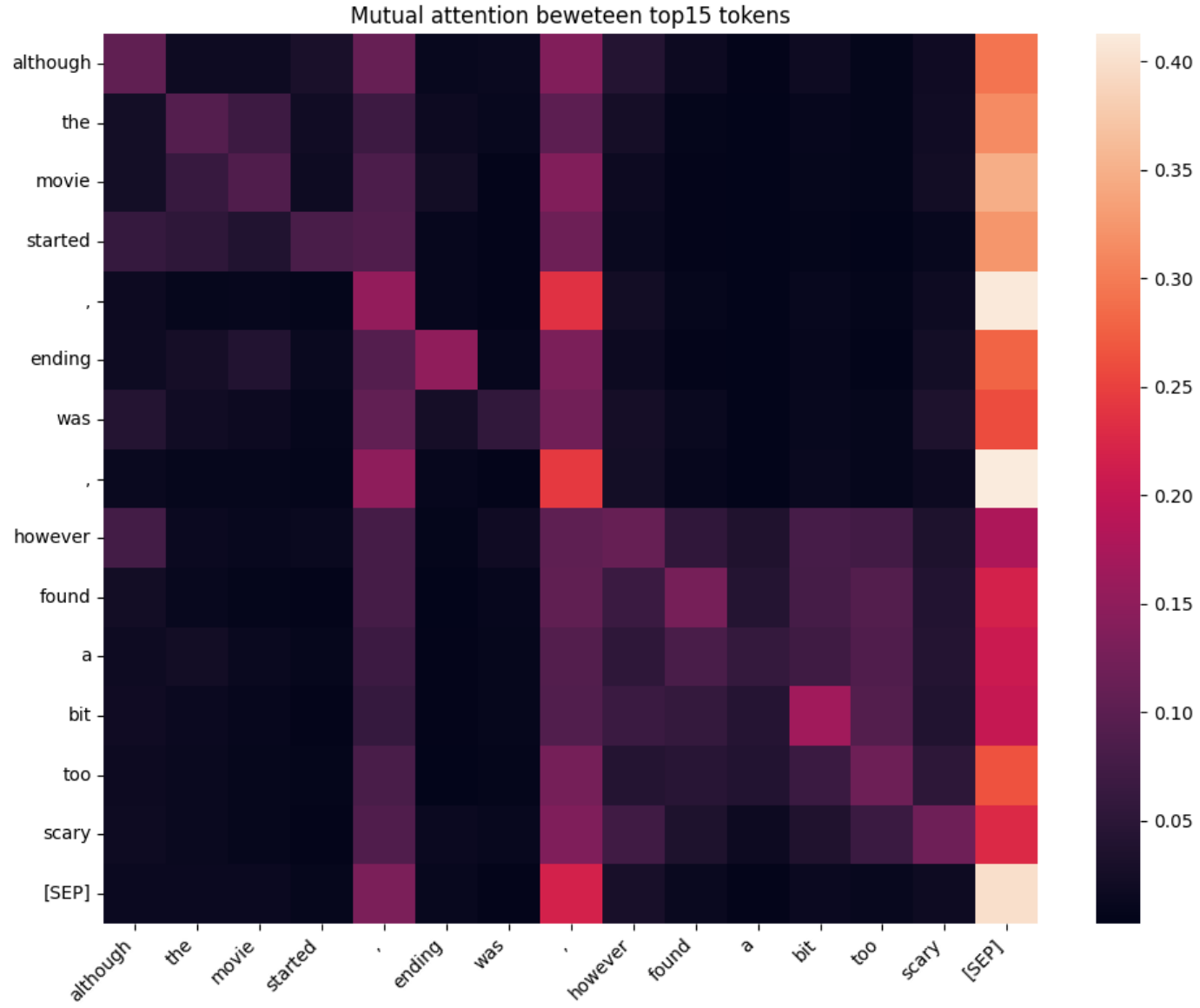
i have to admit thanks to this movie i'm now afraid of mannequins. hahaha.|br /||br /|but yes, first off the acting in this movie at least by my standard is pretty swell. most of the actors are pretty decent in their roles. the script also seems to be pretty good too, sure some cheesy stuff in there but also some decently written character and some damn scary scenes. i STILL get shivers thinking about that one scene with the dude and the mannequins. brrr.|br /||br /|yeah, I'll say you should check this movie out it's pretty good, and very entertaining. a good watch. 8/10

## Attention heatmaps

Since the IMDB texts are quite long, it might be beneficial to show a heatmap of a short sentence, resembling an IMDB review.

In [29]:
```
mock_review = "Although the movie started off slowly, the ending was funny and thrilling
```

In [30]:
```
plot_cls_heatmap(mock_review,model_attention, tokenizer, k=15)
```




There is still the problem of the punctuation marks, but it is clear that related words attend to eachother more.