

# Ambiguity in Humor: Detecting English Puns

Viktoriia Tsosenko  
Computational Linguistics Department  
Heinrich Heine University Düsseldorf  
Viktoriia.Tsosenko@hhu.de

February 9, 2026

## Abstract

Humor is a complex linguistic phenomenon, and puns are one of the good examples of how ambiguity creates amusement. Detecting puns automatically remains a challenging task for natural language processing systems because it requires understanding lexical ambiguity and context. This paper focuses on the automatic detection of homographic puns, which is the first step in computational humor analysis. For this purpose, we used a publicly available, pre-annotated dataset the **SemEval-2017 Task 7**, together with a pretrained BERT-based language model. The BERT model is fine-tuned on the **SemEval** dataset to classify sentences as either containing a pun or not. To achieve this, we used PyTorch for training. The goal of this work is to implement a simple but effective model for pun detection and to get experience with data handling and machine learning. The findings may provide a starting point for more advanced research, such as pun location or interpretation, and highlight the difficulties of modeling humor computationally.

## 1 Introduction

Human communication is difficult to imagine without humor. “If computers are ever going to communicate naturally and effectively with humans, they must be able to use humor.” (Binsted, 2006, p.59). Humor also has its “beneficial aspects, because it banishes sadness and boredom, puts the individual in an optimistic mood and, in general, lightens the load of everyday living.” (Larkin-Galiñanes, 2017, p.4). Making machines *understand* and *produce* humor is therefore beneficial for people. Among the various forms of humor, puns are particularly interesting because they exploit the multiple meanings of words or the similarity of sounds to generate unexpected interpretations. It is known that words often have multiple meanings, and Zipf (1949) indicates that the most frequently used words tend to have more senses than less frequent ones.

Studying puns is not only relevant for humor research, but also useful in real-life applications. Automatic pun detection can improve tasks such as chatbot

interaction, sentiment analysis, and figurative language understanding, where recognizing humor or ambiguous language is essential to respond naturally. This paper focuses on homographic pun detection, which occurs when one and the same word carries multiple meanings in a sentence. The **SemEval-2017 Task 7** dataset provides English sentences labeled as a pun or not a pun. The task represents a binary classification problem: given a sentence, the model must decide whether it contains a pun. The research objectives are as follows:

1. To preprocess and prepare the **SemEval-2017 Task 7** dataset for model training
2. To fine-tune a pre-trained BERT model for the binary classification of sentences as pun or non-pun
3. To evaluate the model's performance using standard metrics such as accuracy and F1-score
4. To analyze the results of transformer-based approaches in understanding humor and ambiguity

## 2 Humor and Ambiguity

The perception of what humor is has undergone major changes throughout the history. Larkin-Galiñanes (2017) believes that humor as we know it today, originates from the 20th century. There is no single, universally accepted definition of humor though. Some writers believe it is even impossible to define what exactly humor, as a term, means (Attardo, 1994).

Plenty of linguistic devices are used to create humor. In Ancient Greece and Rome, Aristotle and other classic writers tried to find out what linguistic techniques were used to produce laughter, and ambiguity was listed among them. In *Rhetorics*, Aristotle talks about “twisting from the proper and apparent sense”, or giving a word “a different ‘turn’” that awakes laughter. He mentions “temporary deception practiced upon the listener”, comparing it with some kind of riddle. When listening to a joke, the listener “does not at once trace the resemblance”, but by “discovering its meaning, the listener learns something new” and is therefore experiencing a satisfying mental “aha!” moment. (Aristotle, Translation, 2009, p.319-320). This is a good evidence that ambiguity in humor has interested people starting from Ancient Times.

The examples below illustrate how ambiguity works at different linguistic levels (morphological, lexical, syntactic etc.), and how humor depends on it.

**Structural ambiguity** Dubinsky presents a joke from the Dilbert cartoon (Dubinsky, 2011, p.56):

*Person A to Person B:* Would you like to buy advertising in my new magazine called Gullible World? We have between one and two

billion readers!

*Some time later, Person A to Person C:* I figured out how to make three readers sound like a lot.

The humorous phrase *one and two billion readers* could mean:

1. [[one and two] billion]
2. [one] and [two billion]

**Compound words** (Dubinsky, 2011, p.44):

1. The word *man*: “while a milkman delivers milk, you really don’t want your garbagemen or firemen bringing garbage or fire”.
2. “In noun + noun compounds, sometimes the first noun tells you what the second one is made of or with, as in *cheesecake* or *apple pie*. Sometimes you hope that it doesn’t, as in *shepherd pie* (more commonly known as *shepherd’s pie*) or *Girl-scout cookies*. Or you find out that it does, and wish you hadn’t, as in *headcheese*.”

**Idioms** (Dubinsky, 2011, p.46), illustrated with an idiom *a piece of me* (meaning “challenge to fight”) and its literal meaning:

1. Frankenstein’s monster gets into an argument at a bar and says: “*You want a piece of me? Huh? Huh? You want a piece of me?*”.

### 3 Classification of Puns

“Punning is a form of humorous wordplay based on semantic ambiguity between two phonologically similar words – the pun and the target – in a context where both meanings are more or less acceptable.” (Palmann, 2025, p.1)

There are different types of puns. The most relevant ones for natural language processing application are homophonic and homographic puns.

**Homophones** Homophonic puns sound the same, but their spelling is different. Dubinsky (2011, p.51) refers to a clear example from a 2001 FoxTrot panel:

1. Jason Fox and his friend Marcus show seven carved pumpkins in a row. On the pumpkins, they carved, in order: the number “3,” a decimal point, then “1,” “4,” “1,” “5,” and “9.” Jason tells his sister: *We’re calling it “pumpkin pi.”*

The joke relies on the homophone pair *pie* (as in pumpkin pie) and *pi* (a mathematical constant).

**Homographs** Homographic puns share the same spelling, as in:

1. I used to be a banker, but I lost *interest*.

Interest can mean curiosity and money earned on savings.

It is important to mention congruity between the two senses when talking about puns. “Nothing is more futile than the irrelevant pun that is based on only a verbal similarity and brings out no contrast, innuendo, or congruity of meaning” (Stanford, 1972, p.72). With no semantic opposition in the punning text, it will not be a joke anymore (Hempelmann et al., 2017). Raskin expressed that puns, based “on purely phonetical and not semantical relations between words” are called “bad puns” (1985, p.116).

Understanding a pun often requires recognizing multiple meanings simultaneously, which is easy and intuitive for humans, but remains difficult for machines. Next chapters are dedicated to the automatic pun detection, namely of homographs.

## 4 Dataset

### 4.1 Dataset Overview

The **SemEval-2017 Task 7** includes six separate files with homographic puns and six with heterographic ones. For this study, only the homographic pun dataset is relevant, which consists of three pairs of files: each pair includes a `.xml` file with sentences and a corresponding `.gold` file with annotations. It is structured as follows:

#### **Subtask 1: Pun Detection**

The `.xml` file contains 2250 annotated sentences, each with its own *sentence ID*. Each token in a sentence also has its own *ID*. The corresponding `.gold` file provides labels for each sentence ID (1 for pun, 0 for no pun).

#### **Subtask 2: Pun Location**

The `.xml` file includes only pun sentences (1607). The `.gold` file specifies the position of the punning word in each sentence by its ID.

#### **Subtask 3: Sense Annotation**

The `.xml` file contains 1298 sentences with puns, where the punning word is annotated with two different WordNet sense IDs, representing its two possible meanings in context. The `.gold` file with these annotations indicates the correct senses.

For this paper, only **Subtask 1: Pun Detection** was used, split into 3 subsets, namely: a training set, a validation set and a test set. It is important to mention what kind of sentences represent the non-pun class in this dataset. Non-Puns from SemEval are mostly proverbs, idioms, or other forms of figurative speech that are often used to create humorous effect:

- (1) **hom\_17**: “You have two choices for dinner: Take it or Leave it.”
- (2) **hom\_57**: “It’s easy to be wise after the event.”
- (3) **hom\_65**: “If all appears to go well, you missed something.”

(1) is an example of a set expression, (2) is a proverb and (3) is a humorous saying. Such sentences are not “normal” declaratives, they are often abstract, full of wisdom and might sound old-fashioned in a way.

## 4.2 Dataset Preparation

Before proceeding with the fine-tuning we had to prepare the dataset, so Bert can understand it. The file `subtask1-homographic-test.xml` is structured as a tree. Consider the following sentence from the dataset:

```
<text id="hom_13">
  <word id="hom_13.1">Diligent</word>
  <word id="hom_13.2">youth</word>
  <word id="hom_13.3">makes</word>
  <word id="hom_13.4">easy</word>
  <word id="hom_13.5">age</word>
  <word id="hom_13.6">.</word>
</text>
```

In the example above, `<text>` elements represent sentences, divided into individual `<word>` elements. Each `<text>` has an ID with the corresponding IDs for its child nodes. To parse the raw `.xml` file, Python’s built-in `xml.etree.ElementTree` library was used. Every individual word node of the same `<text>` was collected and joined together into the full sentence. The parsed sentences were merged afterwards with their labels from the `.gold` file based on the ID.

With 1607 pun- and 643 non-pun sentences, the dataset **Subtask 1: Pun Detection** is imbalanced. To avoid the bias of predicting the majority class, we used class weights for imbalanced data and to maintain the distribution of puns and non-puns across all splits, we added **stratified sampling**. Finally, we split the imbalanced dataset into 70% for training, 15% for validation and 15% for testing.

## 5 Fine-Tuning Preparation

### 5.1 Classifier

The classifier was build using transformers from Hugging Face (`BertTokenizer`, `BertForSequenceClassification`), with the model `bert-base-uncased`.

BERT requires numbers as an input, so the merged sentences from the dataset were tokenized using `BertTokenizer`. The table below illustrates how the model sees and tokenizes a sentence:

Table 1: BERT Input Representation

Tokens:	[CLS]	i	kept	failing	to	get	a	...	cut	[SEP]
IDs:	101	1045	2921	7989	2000	2131	1037	...	3013	102

As seen in Table 3, BERT adds two special tokens to each sentence, namely [CLS] (to store the meaning of the sentence) and [SEP] (a sentence separator). The [CLS] tokens gather values of each sentence that are used in the later step for classification.

Since the model `bert-base-uncased` was pretrained on a large volume of unlabeled English data, it learnt “an inner representation of the English language” through Masked Language Modelling and Next Sentence Prediction. (See, <https://huggingface.co/google-bert/bert-base-uncased>). When combined with the class `BertForSequenceClassification`, BERT is able to perform classification tasks. It contains a linear layer that uses weights from [CLS] tokens and converts them into logits (raw scores for each possible class, e.g. [-1.5, 2.8]). (See, <https://huggingface.co/docs/course/chapter1/5>). This way, the model chooses the highest score between two indexes and matches 0 to a non-pun and 1 to a pun.

## 5.2 Training. Validation. Testing

The three phases were conducted by implementing PyTorch.

Table 2 illustrates what hyperparameters were used.

Hyperparameter	Value
Redproducibility	42
Optimizer	AdamW
Learning Rate	$1 \times 10^{-5}$
Batch Size	16
Epochs	2-4

Table 2: Model Training Hyperparameters

Training was set by the mode `.train()` that activates two layers (dropout and batch normalization). It receives `input_ids`, `attention_mask` and `labels` (passed directly to the model). After each batch, old gradients are removed by `.zero_grad()` and weights are updated with `.step()`.

For validation and testing, the mode `.eval()` was used to deactivate dropout layers and `torch.no_grad()` to avoid gradient calculation, so the model concentrates only on a forward propagation to make a prediction. Similar to training, both validation and testing receive `input_ids`, `attention_mask` and `labels`, but the `labels` for them are not used for predictions and are revealed only afterwards to check their accuracy. The model’s accuracy was calculated with

`torch.argmax(logits, dim=-1)` that matches logits with the highest value for either puns or non-puns.

## 6 Results

### 6.1 Classification Metrics

To evaluate the model’s performance, the `classification_report` function from the `sklearn.metrics` module was used. It includes a text report showing the main classification metrics (such as precision, recall and f1-score) and provides averages to understand overall performance in a better way (namely accuracy, macro average and weighted average).

For reproducibility reasons, the training was conducted with a fixed seed value. The most relevant reports are included in this paper (more reports can be found on our GitHub).

### 6.2 Fine-Tuning with Imbalanced Dataset

The model performed well from the first runs. With some hyperparameters adjusted for each run, the accuracy remained between 90 and 95%.

	precision	recall	f1-score	support
<b>Non-Pun</b>	0.93	0.82	0.87	97
<b>Pun</b>	0.93	0.98	0.95	241
<b>accuracy</b>			<b>0.93</b>	338
<b>macro avg</b>	0.93	0.90	0.91	338
<b>weighted avg</b>	0.93	0.93	0.93	338

Table 3: Classification Report 1 (2 epochs)

The results in Table 3 demonstrate high accuracy. Precision of predicting both puns and non-puns is correct 93% of the time. Recall differs between both classes for about 15%, favoring puns. Since the original SemEval dataset is imbalanced and has much more pun sentences, the model could have learnt that to choose a pun is always a correct guess, and thus, it became biased. To achieve better recall scores for non-puns, the number of examples from the majority class was reduced to match the number of the non-puns.

### 6.3 Fine-Tuning with Balanced Dataset

As a result, the imbalanced SemEval dataset was changed to a balanced one, with 643 sentences for puns and 643 for non-puns. This way the model should learn both classes equally. Several runs were conducted using different hyperparameters to determine the best results.

Report 6 used a 70/15/15 data split, while Report 8 a 60/20/20 split. With the same number of batch size (16) and different number of epochs (3 for Report

	Report 6			Report 8		
	precision	recall	f1-score	precision	recall	f1-score
Non-Pun	0.92	0.86	0.89	0.89	0.91	0.90
Pun	0.86	0.93	0.89	0.90	0.88	0.89
Support	193			258		
<b>Accuracy</b>	<b>0.89</b>			<b>0.90</b>		

Table 4: Model Performance with Different Data Split

6 and 2 for Report 8), both models achieved similar overall accuracy (89% and 90%). However, their performance differs a bit. Recall for Report 6 indicates that 93% of puns were detected correctly, while for the non-puns this score is 7% lower, indicating again a slight bias toward predicting puns. In contrast, Report 8 appears to be more balanced, with recall scores for Puns and Non-Puns remaining closer together (88% and 91% respectively).

As for the validation accuracy, there were some differences as well. In Report 6 it started at 0.9016 and did not change neither for epoch 2 nor for epoch 3, which suggests that the model reached its potential quickly. It was different with Report 8, its validation accuracy increased from 0.8638 to 0.9066, indicating better learning. We also trained the model for a 60/20/20 split with 3 epochs (see classification report 7 on GitHub). Though there was an increase in validation accuracy (from 0.9066 to 0.9144), the model’s performance did not change at all, depicting the same metrics as in Report 7.

Thus, Report 8 indicates that despite the smaller training set, the 60/20/20 split with 2 epochs was a better option for this particular dataset. Its overall accuracy is slightly lower, as in Table 3 (90% vs 93%), but it shows nearly identical precision and recall scores for both classes.

## 7 Diagnostic Evaluation

### 7.1 Control Set

As stated in Dataset Overview, the non-pun examples from SemEval are represented by figurative language. The next step of this paper was to test the classifier on the declaratives that are not part of figurative speech. To achieve this, a new set with 100 samples (50 puns and 50 non-puns) was created. Declarative statements with no *clever* structure, were chosen for the non-pun category, as in the examples below:

- (4) “I need to buy groceries after work.”
- (5) “The project deadline is next Wednesday.”

(4) and (5) are just declarative sentences, with no characteristics of a proverb or an idiom. If BERT learned how to detect a pun, it should be able to classify the new non-puns correctly, even though it was not trained on such examples before.

The diagnostic test was conducted on the same models from Table 4.

	Report 6			Report 8		
	precision	recall	f1-score	precision	recall	f1-score
Non-Pun	0.80	0.16	0.27	0.42	0.10	0.16
Pun	0.53	0.96	0.69	0.49	0.86	0.62
Support	100			100		
<b>Accuracy</b>	<b>0.56</b>			<b>0.48</b>		

Table 5: Diagnostic Evaluation Using a Control Set

The result for the new set was not as expected (Table 5): accuracy sank to 48-56%, depending on the hyperparameters used. Though both models managed to detect most of the puns, they missed more than half of the opposite class, labeling most of the non-puns as puns.

Report 6 was trained on more data than Report 8 (900 samples vs 771 samples, respectively), which led to a higher accuracy of the first one. However, it does not indicate that Report 6 performs better. Both models failed at predicting puns (recall 16% and 10%).

It appears that BERT did not actually learn what puns are, instead, it learnt what belongs to class “0”, namely proverb-like or idiom-like structure and if the sentence did not match the structure, it automatically was labeled as “1”. That is why the new non-pun sentences fell to the category of puns. So it means that Table 3 and Table 4 show high accuracy not because the classifier was actually able to *detect* puns or ambiguity, it was taught to recognize proverbs and idioms instead.

As a solution, we increased the size of the SemEval Dataset by adding 643 non-puns with literal meaning to match the existing 643 proverbs/idioms. The model was then retrained and its performance was evaluated again.

## 7.2 Updating SemEval Dataset

As seen in Table 6, updating the dataset has resulted in a high model’s performance, with non-pun recall rising from 2% to 92%. The overall accuracy increased to 89% in comparison to Table 5.

	precision	recall	f1-score	support
<b>Non-Pun</b>	0.87	0.92	0.89	50
<b>Pun</b>	0.91	0.86	0.89	50
<b>accuracy</b>	<b>0.89</b>			100
<b>macro avg</b>	0.89	0.89	0.89	100
<b>weighted avg</b>	0.89	0.89	0.89	100

Table 6: Control test with updated SemEval Dataset

Some of the misclassified non-puns:

- (6) “The *government* announced new environmental regulations.”
- (7) “He is allergic to *peanuts*.”
- (8) “Many people *commute* to the city for work.”

The words in italics from (6-8) are not used ambiguously, but can be often found in pun sentences. For example, sentences with the same words found in the SemEval dataset (hom\_1913, hom\_820 and hom\_1955) are puns, which explains why sentences (6-8) were misclassified. It also demonstrates that BERT memorized the vocabulary of the training set and what category they belonged to. Thus, when receiving a new sentence containing a word from the training set, BERT makes its prediction based on what it learned before.

Some of the misclassified puns:

- (9) “Why don’t scientists trust atoms? Because they *make up* everything.”
- (10) “Why are skeletons so calm? Because nothing *gets under their skin*.”
- (11) “I’m glad I know sign language, it’s pretty *handy*.”

Some of the puns from the SemEval dataset have a question-answer structure as in (9) and (10) but they were classified wrong by BERT. This means that the model classified them as a pun not because of the pattern recognition, but because it looked for the previously seen vocabulary and failed to find the same words in the training set. Additionally, (10) contains an idiom, that is why it could be also misclassified as a non-pun.

## 8 Conclusion

BERT is a powerful language model that can perform different tasks when used with a classifier. The aim of this paper was to detect ambiguity in humor by training BERT to recognize puns. The fine-tuned model demonstrated high accuracy of 90%-95% but failed the diagnostic test by favoring puns. Updating the SemEval dataset to 1.800 training samples has resulted in a much better performance (92% recall instead of 2%) and a balanced F-1 score for both classes (89%).

However, despite the model’s accuracy of 89%, it would be wrong to say that BERT understood what puns are. Instead, it developed a lexical bias based on the words from the training set. Which proves that detecting humor still remains a significant challenge for machine learning.

## 9 References

1. Aristotle. (2009) The Rhetoric of Aristotle: A Translation. Translated by Edward Meredith Cope, Brittlebooks
2. Attardo, S. (Ed.). (2017). The Routledge Handbook of Language and Humor. Routledge
3. Attardo, S. (1994). Linguistic theories of humor. Mouton de Gruyter
4. Binsted, K., Bergen, B., Coulson, S., Nijholt, A., Stock, O., Strapparava, C., . . . & O'Mara, D.(2006). Computational humor. Special sub-issue IEEE Intelligent Systems
5. Dubinsky S, Holcomb C. (2011). Understanding Language through Humor. Cambridge University Press
6. Hempelmann C, Miller T. (2017) "Puns. Taxonomy and Phonology". The Routledge Handbook of Language and Humor, edited by Attardo, Routledge, pp. 95-107
7. Larkin-Galiñanes, C. (2017). "An Overview of Humor Theory." The Routledge Handbook of Language and Humor, edited by Attardo, Routledge, pp. 4-16 Taylor & Francis
8. Pallmann A., Miller T. (2025). "What's in a pun? Assessing the relationship between phonological distance and perceived funniness of punning jokes"
9. Tristan Miller. (2014). Towards the automatic detection and identification of English puns
10. Zipf, G. K. (1949) Human Behaviour and the Principle of Least Effort: An Introduction to Human Ecology (Addison-Wesley, Cambridge, MA); reprinted in Zipf, G. K. (1972) Human Behaviour and the Principle of Least Effort: An Introduction to Human Ecology (Hafner, New York), 1st ed., pp. 19-55
11. <https://huggingface.co/google-bert/bert-base-uncased>
12. <https://huggingface.co/docs/course/chapter1/5>
13. <https://www.geeksforgeeks.org/deep-learning/what-does-model-train-do-in-pytorch/>
14. [https://docs.pytorch.org/tutorials/recipes/recipes/zeroing\\_out\\_gradients.html](https://docs.pytorch.org/tutorials/recipes/recipes/zeroing_out_gradients.html)
15. <https://user.phil.hhu.de/ehren/deep-learning-in-nlp-summer-2025/>
16. <https://huggingface.co/blog/bert-101#1-what-is-bert-used-for>