No pun intended - Locating English puns

GitHub Repository : https://github.com/DavVratChadha/WhySoPunny

Dav Vrat Chadha, Alfred Xue, Daniel Deza

April 18, 2023

Abstract

Linguistic knowledge has several categories ranging from phonology to semantics but the toughest of all of these to capture with a language model is pragmatics, the study of contexts. A field that tends to focus on pragmatics is humor, specifically puns, and it remains a difficult concept to fully grasp. Puns are a humorous play of words that can be broken down into two types: semantic where the humor steps from a word having several meanings and homophonic which relies on two words sounding the same. This paper presents two fine-tuned large language models and their ensemble capable of identifying where a pun is located within a sentence for both types of puns. The models put forward are well-known LLMs, RoBERTa and DeBERTa, and employ a combination of our two new techniques: context-based custom masking using K-means, and pun POS tag frequency weighting, to together exploit the structure of puns and increase location accuracy. The results are competitive with current models despite using its relatively restricted dataset.

As ultimately the work done in this paper is meant to aid in the generatiom of pun explanations, we compliment our research with an analysis on state-of-the-art generation model: GPT4.

1 Problem Statement

Comedy is often considered the true measure of fluency in a language, this is especially true for jokes relying on word play such as puns as they use language-specific homonyms and semantic ambiguity. Despite it being shown that incorporating humor into second language classrooms is beneficial to both learning and dealing with anxiety [1], existing humor data sets are commonly classification based, falling into the classes of joke, pun, or punchline.

The problem addressed is that of identifying both semantic and homophonic puns. Semantic puns are based on words having several meanings such as "Why did the scarecrow win an award? Because he was outstanding in his field!" where field refers to both a field of knowledge as well as a farmer's field. Meanwhile, homophonic puns rely on words sounding similar such as "Why did the bicycle fall over? Because it was two tired!" here "two tired" referencing the two wheels of a bicycle sounds like "too tired".

2 Prior Work

2.1 *ExPUNations: Augmenting Puns with Keywords and Explanations*

ExPUNations: Augmenting Puns with Keywords and Explanations [2] details the creation and usage of a highly detailed data set used to tune state-of-the art NLP models that detect, classify and explain puns. Throughout the paper the authors highlight the limitations related to a binary labeling of the pre-existing data set as either a joke, pun, or punchline. Notably, they explain that it is insufficient for the models to simply understand and generate humorous text as this does not lead to meaningful learning. To combat this issue, the paper suggests the use of features arrays with metrics of understandability, offensiveness, funniness, whether the text is a joke (binary), joke keywords, and the natural language explanation for each text.

The binary classification of whether the text is a joke is done with fine tuning pretrained models: BERT [3] (a deep bidirectional transformer for NLP), RoBERTa [4], and DeBERTa [5]. For the explanation of the joke, the pun-word is searched for in the data set created above and the explanation from the data set is selected as the input, the paper calls this as the "Gold Explanation". If no gold explanation is available in the data set for the given pun, a negative explanation is randomly sampled and used as input. Using these inputs, a T5 model [6] is fine tuned to generate pun explanations ("Generated explanations"), which are later classified on the basis of correctness by training the ELV [7], an explanation-augmented classification model, with the use of explanations as latent variables.

Their work shows that using gold explanations as the training data improves the performance of existing models, which have been shown in the graph below. Out of all models, RoBERTa [4] and DeBERTa [5] perform the best when negative sampling is used together with gold explanation alongside the ELV model for classification. The accuracy of RoBERTa increases to 81.5% (22.5% gain) when including gold explanations, and DeBERTa increases to 79.5% (17.5% gain), while BERT [3] increases to 77% (14.5% increase). The classification accuracy of ELV in training is 65%.

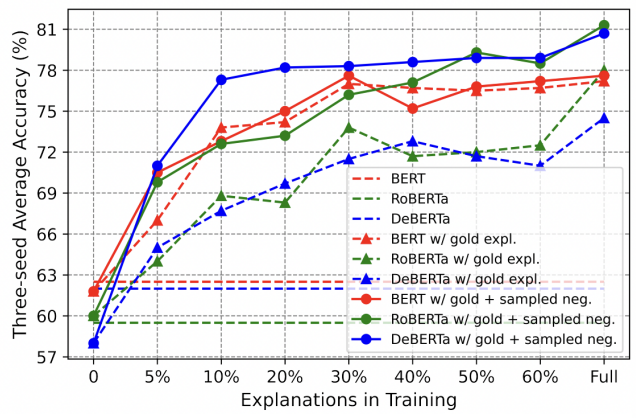

Figure 1: Training accuracy for the models used [2]

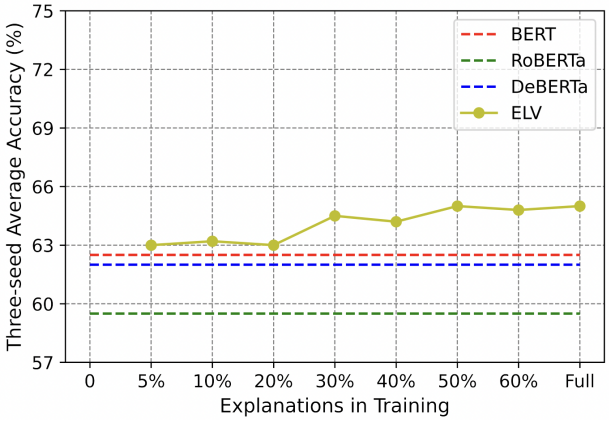

Figure 2: Explanation classification accuracy [2]

Results also show that even though the pun is correctly identified and a relevant explana-

tion is provided, a major limitation in the T5 model is that it generally fails to fully explain phonological similarities in homophonic puns. In other words, when a homophone is used as a pun in a sentence, the generated explanation is often misleading or incorrect.

2.2 *Joint Detection and Location of English Puns*

Joint Detection and Location of English Puns [8] puts forward a new approach of jointly detecting and locating homophonic and semantic puns in a given sentence with the help of a novel tagging scheme. Generally, this problem is solved by using two separate models. The first model detects puns and returns a binary answer. Meanwhile, the second model finds the location of the pun by making single word predictions to check which word in the sentence provides multiple semantic meanings to the text. As a result of using a joint approach with a single model to solve both problems at the same time the error propagation was reduced. With the help of word position knowledge incorporated into a structure prediction model using sequence labeling they manage to capture the structural property of each sentence containing a pun.

2.2.1 Method

The data set consists of sentences in which each word is given a binary tag out of N, P where N means this word is not a pun, while P means the word is a pun. An additional set of tags for each word is also used: B, P, A, where the B tag indicates that the current word appears before the pun, P indicates the word is a pun, and A indicates the word appears after the pun. The model architecture is a Bidirectional LSTM with a concat layer and a CRF (Conditional Random Field) layer. The input to the LSTM layer is word embeddings and binary features for the concat layer. Given a sample from the data set, the model generates a tag sequence as described above. If the input sentence does not contain a pun all words in the sentence are tagged with N and B. BiLSTM captures contextual information in the given sentence, while CRF is set to capture label dependencies at each position using conditional probability [9]. During training, the negative log likelihood of this conditional probability is minimized over all training instances, whereas Viterbi algorithm is applied during testing to search for the optimal label sequence given new input words. The model is trained on two benchmark data sets [10] which comprise of a total of nearly 2900 homophonic and semantic puns. A 10-fold cross validation is applied to get rid of any possible bias.

For training, word embeddings are initialized with a 100-dimensional Glove [11] which are semantic vectors used for word representations. Stochastic gradient descent with a learning rate of 0.015 is used for this process. During testing if the predicted tags contain at least one P, the output of the LSTM layer is regarded as True, i.e. the sentence contains a pun. For the pun location, if the predicted pun is labeled as P in the training data set, the located word is regarded as true.

2.2.2 Result

As compared to the Stanford POS tagger, n-grams, and label transitions, the new NP and BPA tagging schemes yield significantly better scores for pun detection, and competitive scores for pun location, which are shown in the figure below.

System	Semantic Puns						Homoph		
	Detection			Location			Detection		
	P.	*R.*	F_1	*P.*	*R.*	F_1	*P.*	*R.*	F_1
Pedersen (2017)	78.32	87.24	82.54	44.00	44.00	44.00	73.99	86.62	68.71
Pramanick and Das (2017)	72.51	90.79	68.84	33.48	33.48	33.48	73.67	94.02	71.74
Mikhalkova and Karyakin (2017)	79.93	73.37	67.82	32.79	32.79	32.79	75.80	59.40	57.47
Vadehra (2017)	68.38	47.23	46.71	34.10	34.10	34.10	65.23	41.78	42.53
Indurthi and Oota (2017)	90.24	89.70	85.33	52.15	52.15	52.15	-	-	-
Vechtomova (2017)	-	-	-	65.26	65.21	65.23	-	-	-
Cai et al. (2018)	-	-	-	81.50	74.70	78.00	-	-	-
CRF	87.21	64.09	73.89	**86.31**	55.32	67.43	**89.56**	70.94	79.17
Ours – NP	89.19	86.25	87.69	82.11	70.82	76.04	85.33	90.64	87.91
Ours – BPA	89.24	92.28	91.04	83.55	**77.10**	**80.19**	84.62	**95.20**	89.60
Ours – BPA-p	**91.25**	**93.28**	**92.19**	82.06	76.54	79.20	86.67	93.08	**89.76**
Pipeline	-	-	-	67.70	67.70	67.70	-	-	-

Figure 3: Results for different models for pun detection and location in the two benchmark datatset. (P: Precision, R: Recall, F1: Score)[8]

A qualitative analysis of the outputs shows that the model sometimes results in three major types of errors:

1. Low word coverage: Despite using one of the largest pun datasets, the corpus used to train the model is still rather small and thus new words are often mislabeled by the model and not correctly identified as puns. This contributes to 40% of all errors.

2. Detection Error: Although BPA tagging improves model performance, 40% of incorrectly detected punny sentences occur due to improper tagging.

3. Short Sentences: If the input sentence is short, the model is not able to fully grasp the semantic context of all words in the sentence and thus cannot correctly detect the pun.

3 Further Work

3.1 *SemEval-2017 Task 7: Detection and Interpretation of English Puns*

SemEval-2017 Task 7: Detection and Interpretation of English Puns [10] tests and demonstrates several language models for 3 subtasks: pun detection, pun location, and pun interpretation. Models tested include BuzzSaw, Duluth, PunFields, JU_CSE_NLP, and more. The models are tested over two data sets that together contain a total of 2800+ homophonic and semantic puns. Their work is relevant to our project since it tests several pre-existing pun detection and location models. This knowledge would speed up our work, as we can use an ensemble of these models and fine tune them for our purposes. The paper also provides access to a large data set of puns which is useful for training our model.

3.1.1 *e-SNLI: Natural Language Inference with Natural Language Explanations*

e-SNLI: Natural Language Inference with Natural Language Explanations [12] demonstrates the work done to improve the Stanford Natural Language Inference data set by adding a new layer of explanations in natural language done by human-annotation. This new corpus

is referred to as e-SNLI. InferSent model architecture, which includes a BiLSTM with max-pooling layer, is applied to the data set. When given a new sentence, the model aims to generate a relevant explanation for it. The relevance of this paper lies in how it describes how to use a model to generate explanations of a given sentence, which could be repurposed to generate explanations for puns as inputs to the model. This can potentially improve the performance of our model when trained and fine tuned on a data set of puns.

4 Collecting Data

4.1 Data collection pipeline

This is the data collection process that we used to collect our initial pun dataset, mainly consisting of puns gathered from images in reddit posts.

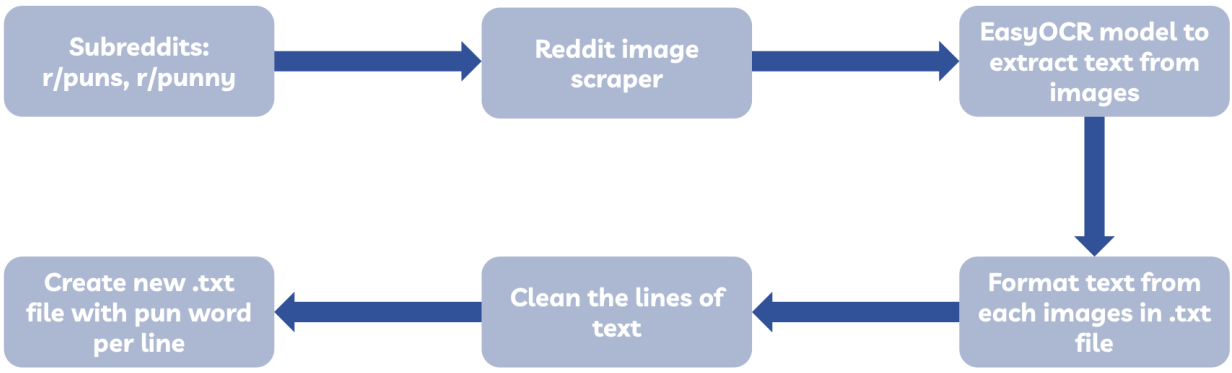

Figure 4: Data collection process

4.2 Reddit: r/puns, r/punny

The data source that was used to collect these puns is the social media platform Reddit. The main reason for this choice was due to the nature of the platform, making it easy to find one topic for a single webpage, in this case, puns. Each "subreddit" is its own community where people can post about a singular topic, and in the r/puns and r/punny subreddits, people post puns. This makes it an easy website to scrape for images. Note that a large number of the puns that people post to the subreddit are located within images, so this needs to be solved.

4.3 Reddit image scraper

Since Reddit is a popular website, it contains a lot of data which is useful for all types of purposes. Therefore, finding and coding an image scraper which could save posts from a subreddit was well documented and not very difficult. Essentially, the code (see RedditImageScraper/SubDownload.py in the Github) made use of reddit development tools and tokens in order to collect data. You can also specify how many images to scrape based on what category, i.e. top posts, hot posts, new posts, etc. This resulted in a large number of images which were not duplicates, but we needed some way to get this image format into text.

4.4 Optical Character Recognition (OCR) model

An OCR model (see easyOCR.py) essentially reads images in and can give the location of any text as well as the string format itself. This string format is required to feed into our large language models as we cannot use the image itself. Every output was written to a new line in our first .txt file which contains all of the full puns.

4.5 Formatting and cleaning

One main concern with this process is that using this OCR model meant that any image providing context to the pun would be removed and only the text would be left. Therefore, we could not use any puns which had image context. For example, in the poor image example below, the image provides context to the pun and therefore cannot be used as the explanation would not make sense without the image. In the good image example, we can see that the pun does not rely on any context from the image and the image is only used for the format of the pun itself. This was the main cleaning process for the images.

Figure 5: Examples of bad (visual reliant) and good (text reliant) data [13]

For cleaning and formatting the .txt file, the main process was to get rid of unwanted characters which were picked up by the OCR model. This includes misidentified characters (e.g. I and —) and words rearranged due to the placement in the image (puns have different text formatting). This cleaning/formatting was done through a manual check.

4.6 Pun word .txt file

Lastly, we need another .txt file which contains the pun word for each corresponding pun in the full puns .txt file. The pun word was written manually to match each line in the first .txt file and both of these text files were put into another script which formatted the puns into the same format as the SemEval dataset so we would maintain the same format (xml trees).

5 Locating puns

Before exploring the details of the model, the figure below is a graphical representation of our model. Each part will be further explained in the subsequent sections.

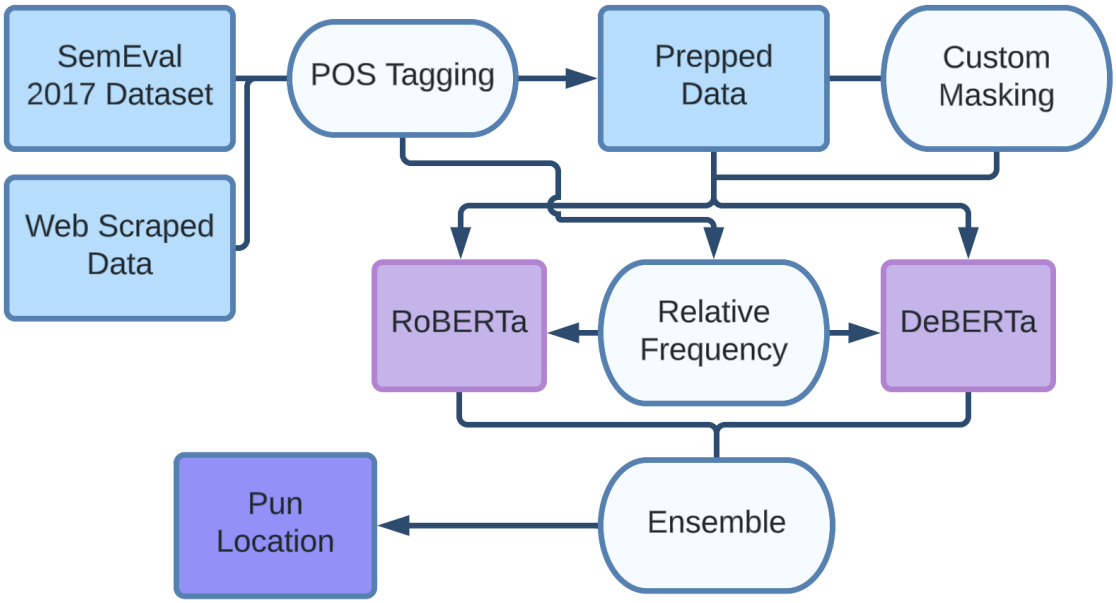

Figure 6: High level graphical representation of the Why so Punny model

5.1 Preprocessing the data

On top of the web-scraped data we also make use of the SemEval dataset [10]. This dataset is a collection of .gold and .xml files, which contain sentences, pun locations, pun labels, negative samples, and much more very specifically formatted for the SemEval 2017 competition. This format is very different from what we desire to train our models on, meaning the data needs to be preprocessed before it can be passed into our models. The first step in this process is converting the xml trees into a list of sentences. Labels for negative sample sentences, type of pun, and pun locations for remaining sentences are extracted from the .gold files, allowing us to separate the sentences and label them accordingly. We also POS tag all sentences and puns since

1. using the lemma* instead of the original word reduces the overall complexity of the models, decreasing entropy for the classifier and making pun location with sequence classifiers easier

2. we believed that certain POS tags for words in the English language would have a higher chance of showing up as puns than others

* a lemma is a simplification of a word, for example the lemma of "playing" is "play".

POS tagging classifies each word according to its parts of speech (noun, adjective, adverb...) and is accomplished with the spaCy library in python using en_core_web_lg. We calculated the frequency for each tag showing up as a pun in our dataset and noticed a trend

with what type of words were more likely to be puns, this is visualized in Figure 7. To use this as a feature we compute the Hadamard product of the likelihood tensor (output of the models for classifying each word as pun/not pun) with a relative frequency heuristic tensor where the relative frequencies correspond to the tag of that input word. We use the element-wise product of these two matrices to calculate the accuracy of the model. To combat the issue where a type of word does not appear as a pun in the test set, leading to a 0 relative frequency, we apply Laplace smoothing before calculating the relative frequency heuristic values. Laplace smoothing ensures that no weight is 0 according to the following formula:

$$\text{relative frequency} = \frac{\text{number of times a tag appears as a pun} + 1}{\text{total number of puns in the dataset} + \text{number of distinct tags in dataset}} \tag{1}$$

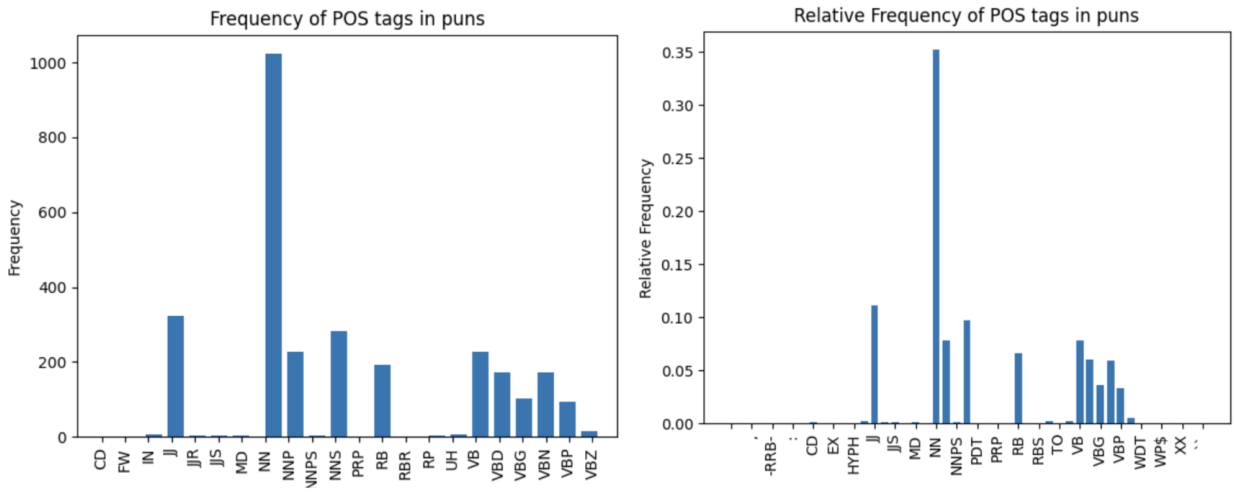

Figure 7: Frequency and Relative frequency (Laplace smoothed) for training set

5.2 Custom Attention Masking with K-Means Clustering

5.2.1 Dynamic Random Masking

We use masking to reduce the pool of words that could be the pun in a given sentence by setting the probability of that word being the pun to zero. The task now becomes how do you chose which words to mask. While there are default masking techniques for RoBERTa and DeBERTa are to randomly mask 15% of the tokens, we found their impact on the performance to be negligible if not non-existing. Other masking techniques including Span Masking, involving masking a contiguous span of tokens instead of individual ones, often lead to the pun being masked, causing the model to not pay attention to it, and thus underperform. This indicates that we require a new making technique that masks tokens based on context and ensures the pun does not get masked out.

5.2.2 Contextual Masks using K-means

To solve the issues caused by default masking techniques while also increasing model performance, our models apply masking based on k-means clustering. Each word in the sentence

has a token ID obtained after passing the sentence through the encoder. Words with similar contexts will have the same/similar token ID. Based on the token IDs, we use the unsupervised learning algorithm of k-means clustering to cluster the words with similar token IDs. The choice of max number clusters follows the relation:

$$k = max(\lceil \text{length of sentence} * \alpha \rceil, min(4, \text{length of sentence})) \tag{2}$$

Where α is a hyperparameter less than or equal to one. For DeBERTa and RoBERTa α is set to 1 and 0.5 respectively. Note that since the clusters are randomly initialized leads to the possibility of empty clusters, therefore k is an upper bound on the number of clusters. Below is an example of the k-means clustering performed on the sentence "Did you hear about the crime that happened in a parking garage? It was wrong on so many levels." where the pun is on "levels".

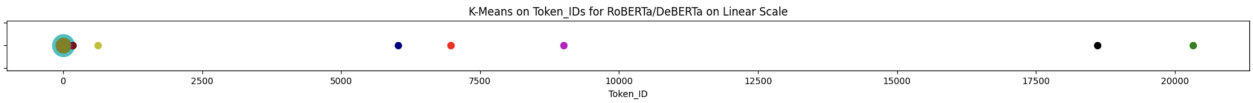

Figure 8: 1 Dimensional K-means performed on the sentence "Did you hear about the crime that happened in a parking garage? It was wrong on so many levels."

The results are that for 95.91% of the sentence in our dataset, the pun word is in one of two clusters:

1. the cluster with the most amount of words
2. the non-empty cluster with the fewest amount of words (generally cluster size = 1)

The first case happens when the pun is semantic, this occurs because the way semantic puns are structured is that it has many supporting words that give context to the pun. Hence the pun appears in the cluster with the most words, such is the case in the above figure, the pun word "levels" was found in the large cyan cluster. On the other hand, when the pun is homophonic it has a different context than the rest of the words and is often isolated in its own cluster or cluster size = 1. Furthermore, 2% of pun words are in the second smallest cluster.

Once the clusters are identified we mask out the cluster we are not interested in by setting the masking value for them to 0 while the others are set to 1, giving us our context-based attention masks. One consequence of this masking is that our model's accuracy is capped at our capture rate = 95.91% for roBERTa and about 97% for DeBERTa as for the remaining time the pun word is masked out meaning it cannot be predicted correctly.

5.3 Fine-Tuning the Models

Our models, RoBERTa and DeBERTa, take the original sentences, tokenize them with their respective tokenizers, and apply custom attention masks to ensure the models do not give attention to irrelevant words. These input sentences are also right-padded by the models to ensure all sentences have the same length. For our models, we set this length to be equal to the length of the list of words containing the largest joke plus the start of sentence (SOS)

and end of sentence (EOS) token (which brings max_len to 79 in our dataset). To further fine-tune these pre-trained models we made several modifications:

Scheduler: The scheduler exponentially decreases the learning rate as the model progresses training over the number of epochs.

Early Stopping: Following this to deal with overfitting we implemented early stopping with a sliding window of 15 epochs, that is if an epoch's validation loss is higher than the previous 15, then the model is saved and training is stopped. This allows us to set the number of epochs to 200 and let the model learn to train until it has not fully learned the data. Note: the models do not train for the entirety of 200 epochs. They stop much earlier due to the early stopping window.

Loss function: The loss function used to calculate the loss during training is the Cross-Entropy Loss. This is because the model is being fine-tuned for sequence classification using the RoBERTaForSequenceClassification class, which has a classification layer added on top of the RoBERTa model. The RoBERTaForSequenceClassification class uses Cross-Entropy Loss as the default loss function for sequence classification tasks, and we are using it. The same is done for the DeBERTa model.

Optimizer: We are using NAdam optimizer because of its accelerated convergence and robustness to noisy data. The learning rate for each model is different; 10^{-5} and $1.9 * 10^{-5}$ for RoBerta and DeBerta respectively.

Grid Search: For tuning the hyperparameters, we use the technique of grid search. Starting by only tuning one hyperparameter, and when that is done, we fix this value, and start converging the next hyperparameter. We started with tuning the optimizer, then learning rate, initially using a constant learning rate, but soon shifting to an LR-scheduler, followed by using early stopping with sliding window of size 15 and number of epochs set to 200. We also tried different loss functions and gradient clipping, but settled with the Cross Entropy. Lastly, we fine tunes the multiplier for number of clusters for contextual masking.

5.4 Ensemble

To combine the predictions of the RoBERTa and DeBERTa models we use an ensemble, specifically a weighted voting ensemble. Both the RoBERTa and DeBERTa are trained on the same data independently and vote on the correct pun words. These votes are weighted by the model's confidence rate in that output. To compute the confidence rate, we take the softmax of the logits for each model, take their Hadamard product with relative frequency heuristic, and then normalize the new tensor. We apply argmax over this new tensor to find the word with the highest confidence rate to be the pun. The figure below shows the voting power or weighting, each model is given to maximize accuracy. These values were found after running the models on the validation sets with different weights.

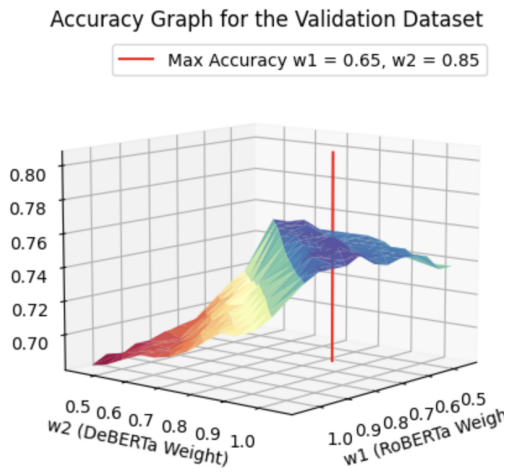

Figure 9: Accuracy as a function of model voting power

5.5 Results

The table below summarizes the accuracies of the models throughout the different versions of the model.

	Additions to base model									
	None			Rel. Freq			Rel. Freq + Masking			Ensemble
Model	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test	Test
RoBERTa	82.31	56.47	58.10	79.16	67.32	65.82	95.65	71.10	67.21	**75.58**
DeBERTa	78.29	55.24	48.84	86.01	68.47	67.92	96.02	75.93	74.07	

Table 1: Accuracy values of our fine-tuned LLMs

We note a rather large accuracy gap between the the training and validation sets, this has to do with our optimizer: NAdam. This optimizer prioritizes converging rapidly sing momentum thus it is not able to fully learn on the data with few epochs. The final accuracy of the model is **75.58%**. This accuracy is competitive with previous state-of-the-art models such as *Joint Detection and Location of English Puns*' 79.20% accuracy. Even more pressing is the model's comparison to GPT-4. In the analysis below we approximate GPT-4's accuracy on pun locating to be 82.77%. Considering the relative simplicity of our model compared to GPT-4 (100M (DeBerta Base)and 125M (RoBerta Base) parameters [17][18] compared to GPT-3's 175B [] which pales in comparison to GPT-4) as well as the much smaller data set we consider this to be an extremely promising result.

6 GPT-4: Location and Generation

6.1 What constitues a good explanation

For this task, we decided that we would use a state of the art generation model, GPT-4, to generate the explanations of what makes the pun funny, given a sentence containing the

pun. First we need to come up with a baseline of what makes a good explanation. For this, we came up with 3 things that the output must contain to show that a model is able to generate explanations: the pun word, the intended word, and the context/pragmatics of the pun. Being a great generation model, GPT-4 would be capable of taking these three inputs and generating explanations, but if the explanation does not contain these three pieces of information, we conclude that the model failed the task.

6.2 GPT-4's locating performance

We passed 1271 sentences through the GPT-4 with a very specifically engineered query prompt: "Give the following sentences, locate the pun, give the intended word, and explain the context of the pun. Return the output in the following format: #" I' m halfway up a mountain," Tom alleged. # The pun is based on the word "alleged" sounding like "a ledge" in the context of being halfway up a mountain. Where the original sentence is included in two #'s followed by a one line highlighting the pun word, intended word and context." We then used the Regex library in python to compare the GPT-4 output with the ground truth and it was found that that GPT-4 accuracy to locate puns is 82.769%, not too far from our models. The model fails to recognize the correct pun in 219 sentences out of 1271 we gave it.

Some common cases of failure included the model not being able to understand the context of the sentence, and the model not being able to strictly follow the prompt. One instance of these is shown below (line 1003 in GPT_4_output.txt):

A third - generation clothes designer had it in her jeans. # The pun is based on the phrase "had it in her genes" sounding like "had it in her jeans" in the context of a clothes designer.

Here, the pun word is jeans, but instead of tagging that word as a pun word, the model decided to go against the query and instead return a pun phrase. Even though the output phrase contains 'jeans', the model classifies 5 words as pun, instead of 1, which highlights the ambiguity in model's output and thus we consider it as a failed test case other cases of failure include #Those who live beyond their means should act their wage. #The pun is based on the phrase "act your age" being changed to "act your wage". where the model fails to return the context.

Furthermore, GPT-4 is not the only solution to the problem of explanation generation. For future work, one can take this dataset of 1271 sentence generated by GPT-4, manually check and correct all of them, and then fine-tune a generation model like T5 to generate a sentence for explanation given the pun word, intended word, and context. The first requirement of locating the pun word is satisfied by our ensemble model. Whereas the remaining two can be found by taking the same sentence, masking the pun word, and then passing the alternate definitions for the same/similar word from WordNet, the largest lexical database of English, into our fine-tuned LLMs without the last classifier layer.

7 Detection API: PunPal API

The API to detect puns is available here. We have set up the file for your ease. Simply run it as stated in the file, give it an input sentence, and you will receive an output from all our models with their respective confidence.

We can also evaluate the performance of our pun prediction model on inputs that it was not specifically designed for, such as sentences with no puns, regular jokes without puns, and noisy data including Hapax Legomenon (which includes gibberish) and words that are not present in the language model (e.g. "sadfsdf"). Note: Hapax Legomenon are words that appear only once in the context of the model. So, if a word appears as a pun in a sentence that only appeared once in our training dataset, the model is somewhat likely to label is incorrectly.

In most of these situations, the model performs poorly as it was not trained to handle any of these data inputs, mainly because there is no negative sampling in the training dataset and therefore will not detect if the input is a pun or not (just assumes that the input is a pun). Furthermore, if a non-dictionary word gets added into a pun sentence, the API will perform much worse, even if it is just one random word. This situation makes the model predict the wrong word as the pun word or lowers the confidence of the correct pun word.

8 Impact Statement

While it may seem that a pun locator has rather little broader impact it is nonetheless important to explore the possible implications of this work.

8.1 Affect on ML Applications

Humor is often a hard concept for machine learning models to master, as mentioned previously even models such as ChatGPT can struggle with this. Our model attempts at least simplify the task by narrowing down which words a generative model would have try to relate. This can help improve sentiment analysis, providing a more nuanced understanding of the intended meaning behind a given text. This is also a huge step in improving large language models with newer techniques to capture the pragmatics of the sentence in the English language.

8.2 Social Implications

The most common ethical concern when working with LLMs is that of data privacy as they are trained on internet-scrapped data. Yet this concern is not the most applicable to our application. The main concern when using this data-scrapped LLM is that when a user queries the model its output may reveal someone's private information. Yet our model's output is the location of a pun and therefore extracting private information from the output is rather far-fetched. Furthermore, even if this was a concern in our application it would be no different than Google which few would argue is unethical.

When working with a model trying to understand humor there is often legitimate concern that the model might learn offensive or off-color humor. That being said this is not a generative model meaning the output comes directly from the input. The output in itself cannot be more offensive than the input. Even considering a possible bias in incorrectly

predicting a word to be a pun this not only seems rather tame but also should be rather rare as our custom masking only allows relevant words to be considered.

On the more positive side, the model could be a useful tool for aiding new English speakers get a grasp on humor in another language which, similarly to machines, is one of the hardest frontiers when learning a new language as it requires you to make otherwise unnatural connections between words.

8.3 Further Initiatives to Improve Societal Outcomes

Considering the rather tame societal outcomes an appropriate measure to take would be a filter that restricts our model from taking offensive words as inputs. Furthermore, as in the prior work section, it is possible the teach the model to recognize offensive which could be used to avoid the model being used on puns deemed offensive if during the prediction it passed a certain threshold of offensiveness.

9 Conclusion and Future Work

Our project fine-tunes existing RoBERTa and DeBERTa models to increase pun location accuracy. These changes include using Laplace smoothing, POS tagging, relative frequency heuristics, contextual masking, an ensemble, and more. With these additions we reach a test accuracy of 75.58% on all types of puns, this accuracy rivals the previously achieved 79.20% achieved by the authors of *Joint Detection and Location of English Puns* despite our relatively limited dataset.

To further push this model's accuracy there are several steps we can take. The most obvious is to scrap more data yet this is currently not feasible due to the inefficiency of the web scrapper. Moreover, the current model is trained on both semantic and homophonic puns, training two models separately (one for each type) could very well improve the accuracy. Furthermore, the current model is not trained on negative samples (sentences not containing puns). Training it with negative samples would allow the model to correctly classify non-pun sentences as well as possibly improve accuracy on pun sentences as part of distinguishing pun and non-pun sentences is detecting a pun with confidence. Finally, it may be interesting to further improve the location model using floating point values between 0 and 1 (both inclusive) as this would lead to partial masking of certain words, allowing us to control how much attention is given to words of different context clusters after k-means.

10 References

[1] R. Aboudan, "Laugh and Learn: Humor and Learning a Second Language," International Journal of Arts and Sciences, 2009.
[2] J. Sun, S. Oraby, and A. Cervone, "ExPUNations: Augmenting Puns with Keywords and Explanations," Oct. 2022.
[3] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Association for Computational Linguistics, 2019.
[4] Y. Liu, M. Ott, and N. Goyal, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," Jul. 2019.

[5] P. He, X. Liu, J. Gao, and W. Chen, "DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION," ICLR, 2021.
[6] C. Raffel, N. Shazeer, A. Roberts, and K. Lee, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," JMLR, 2020.
[7] W. Zhou, J. Hu, and H. Zhang, "Towards Interpretable Natural Language Understanding with Explanations as Latent Variables."
[8] Y. Zou and W. Lu, "Joint Detection and Location of English Puns."
[9] Empower Sequence Labeling with Task-Aware Neural Language Model
[10] T. Miller, C. Hempelmann, and I. Gurevych, "Semeval-2017 task 7: Detection and interpretation of English puns," Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), 2017.
[11] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.
[12] O.-M. Camburu, T. Rocktäschel, T. Lukasiewicz , and P. Blunsom, "e-SNLI: Natural Language Inference with Natural Language Explan," NeurIPS, Dec. 2018.
[13] Reddit https://www.reddit.com/r/puns/comments/121h7u9/lifes_a_beach/[Accessed: 27-Mar-2023].
[14] A. Roberts; ProcessingNLP, D. L. M. I. N. L. (n.d.). Exploring transfer learning with T5: The text-to-text transfer transformer. – Google AI Blog. Retrieved March 27, 2023, from https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html
[15] Yang, Z., amp; Carbonell, J. (2020). XLNet: Generalized Autoregressive Pretraining for Language Understanding.
[16] Shirish Keskar, N., amp; McCann, B. (2019). CTRL: A Conditional Transformer Language Model for Controllable Generation.
[17] Facebook Research RoBerta GitHub
https://github.com/facebookresearch/fairseq/tree/main/examples/roberta
[18] Facebook Research DeBerta GitHub https://github.com/microsoft/DeBERTa
[19]About Kindra Cooper Kindra Cooper is a content writer at Springboard. She has worked as a journalist and content marketer in the US and Indonesia. (2022, October 27). OpenAI GPT-3: Everything you need to know. Springboard Blog. Retrieved April 14, 2023, from https://www.springboard.com/blog/data-science/machine-learning-gpt-3-open-ai/