CLEF 2024 JOKER Tasks 1-3: Humour identification and classification*

Notebook for the JOKER Lab at CLEF 2024

Rowan Mann[1], Tomislav Mikulandric[2]

[1]Christian-Albrechts-Universität zu Kiel (CAU), Christian-Albrechts-Platz 4, 24118 Kiel
[2]The University of Split, Ul. Ruđera Boškovića 31, 21000, Split, Croatia

Abstract

The CLEF 2024 JOKER track focuses on the automatic processing of wordplay through three tasks: humour-aware information retrieval, humour classification by genre and technique, and translation of puns from English to French. Recent advancements in Large Language Models (LLMs) have enhanced their conversational abilities, yet they struggle with humour detection and generation. Addressing this gap can significantly improve human-computer interactions.

For Task 1, we implemented a TF-IDF vectorizer and logistic regression model to identify and rank humorous sentences. The model achieved an F1 score of 0.93 for non-puns and 0.73 for puns, indicating robust performance in humour detection. In Task 2, we classified jokes into five categories using logistic regression, Naive Bayes, and support vector machines. The SVM model performed best, with F1 scores ranging from 0.14 to 0.61, showing particular efficacy in classifying wit. Task 3 involved translating puns using the MarianMT model from the Hugging Face library. Although successful, the process was time-intensive, suggesting the need for more efficient methods.

Overall, our approaches demonstrated effective humour identification and translation capabilities but faced challenges in genre-specific classification. This research underscores the importance of improving LLMs' humour processing abilities for better human-computer interaction.

Keywords

LLMs, Humour identification, Humour classification, Large Language Models, TF-IDF

CEUR-WS.org/Vol-3740/paper-175.pdf

1. Introduction

In recent years, Large Language Models (LLMs) have increased in their capabilities exponentially. Since the release of ChatGPT-4 in 2023, the world has quickly become accustomed to interacting with LLMs via a chatbot API that made users feel like they were truly chatting with the machine. A 2024 study applied the Turing Test to participants using ChatGPT-4, finding that humans incorrectly judged ChatGPT-4 to be human 54 percent of the time, showing just how convincingly LLMs can now emulate human conversation. [1]

As impressive as this is, LLMs are still found to be lacking in several areas, humour being one of them. Large language models struggle to reliably detect and explain humour[2][3][4], and generate novel jokes [5]. For humans, humour plays a central role in forming relationships and can enhance performance and motivation.[6] Therefore, giving LLMs a good grasp of humour has the potential to massively boost the success of human-computer interactions.

Puns are a form of humour based on wordplay. Usually puns exploit double meanings of words or similarity of sounds between different words to create a humorous or witty effect, with frequent use of double entendré, homophones, or similar-sounding words. Words which could be used to form puns are words like "profit" and "prophet", for example, or "check" and "Czech".

CLEF 2024: Conference and Labs of the Evaluation Forum, September 09–12, 2024, Grenoble, France
* CLEF 2024 JOKER Tasks 1-3: Humour identification and classification
*Corresponding author.
†These authors contributed equally.
rowanmann93@gmail.com (R. Mann); tomislav.mikulandric@gmail.com (T. Mikulandric)
© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

Perhaps what makes humour analysis such a challenge for LLMs is the many different subtle forms of it that exist. Humour can be "on-the-nose", physical, awkward, subtle, obvious, visual, childish, or intelligent; it's especially hard to define, which presents problems for LLMs.

Wordplay, by its very nature, exploits the intrinsic structure of the source language used and certain characteristics used that may be impossible or difficult to find replacements or analogues of in the target language.[7] Therefore, translating jokes from language to language requires more than simply replacing one word with the corresponding word in the target language. The deeper understanding of context required is an area where LLMs could excel, if methods are developed.

The JOKER track of CLEF 2024 aims to develop interdisciplinary approaches to the automatic processing of wordplay. [8] This year, the JOKER track is split into three tasks:

- Task 1: Humour-aware information retrieval.
- Task 2: Humour classification according to genre and technique.
- Task 3: Translation of puns from EN to FR.

Can LLMs succeed in identifying, classifying, and translating humour? In this paper we will explore this question, ahead we detail our workflow and results for tackling the three tasks of the JOKER track.

2. Task 1: Experimental Setup

2.1. Data Description

The data provided consisted of four JSON files. There was a "corpus" file containing a list of 61,268 pun and non-pun sentences, "queries test" and "queries train" files that list the corresponding keyword linked to the sentences, and a "qrels train" file that if the sentence was a pun ("1") or not ("0").

2.2. Method

Our first step was to merge the data, creating a table with five columns: "qid", "docid", "qrel", "text, and "query". We then used a TF IDF vectorizer to train the model on all our text and the corresponding "qrel" values.

```
#query text and joke text into a single column - TF-IDF Vectorizer
data merged['text all'] = data merged['query'] + " " + data merged
    ['text']


# Fit and transform the combined text
tfidf matrix = tfidf vectorizer.fit transform(data merged['text all
    '])
```

Then we created a logistic regression model based off of our training data and used the model to make predictions based off of our "queries test" data.

```
from sklearn.linear model import LogisticRegression


# Logistic Regression model
model = LogisticRegression()


# Trained model
trained model = model.fit(X train, y train)
```

The model returned relevance scores for each joke in the corpus for each query. (Appendix A)

Based on these results we produced a JSON file listing the "best" or rather, most relevant, jokes. (Appendix B)

3. Task 1: Experimental Results

The results obtained from our model were very positive. Calculation of an F1 score for our model produced results of 0.93 for the "0" (no pun) class, and 0.73 for the "1" class. This indicates that it performed extremely well, with high precision and recall, for identifying sentences without puns and moderately well for identifying those with puns.

4. Task 2: Experimental Setup

4.1. Data Description

The data provided consisted of 3 JSON files. "classification test data" and "classification train input data" each contained "text" column containing thousands of jokes. The classification data was contained in the file "classification train qrels data", which provided the labels for our training data, categorising each joke into one of five classes, IR (irony), SC (sarcasm), EX (exaggeration), AID (incongruity), SD (self-deprecating), WS (wit).

4.2. Method

We merged the training data JSONs to create a dataframe with the text and classes side by side. We preprocessed our text by removing contractions, making all letters lowercase, removing special characters and URLs, then replaced the original text with our cleaned text in our data frame containing the training labels. (Appendix C)

```
prompt terms="""
    You are a robot that ONLY outputs JSON.
    You reply in JSON format with the field 'terms'.
    You provide ONLY semicolon-separated  list of MAXIMUM 3
       scientific terms of a source sentence ONLY.
    You DO NOT add 'Sure, Here are the scientific terms of your
       sentence:'.
    Example source sentence: In the modern era of automation and
       robotics, \
    autonomous vehicles are currently the focus of academic and
       industrial research.? \
    Example answer: {'terms': 'robotics; autonomous vehicles'}
    Now here is my sentence:
"""
```

We encoded our classifications to numbers and used TF-IDF to vectorise the text. The data was then used to train a logistic regression model, a naive bayes model and a support vector machines model.

```
# Train Logistic Regression model
logistic regression model = LogisticRegression(max iter=1000)
logistic regression model.fit(X train tfidf, y train)

# Train Naive Bayes model
naive bayes model = MultinomialNB()
naive bayes model.fit(X train tfidf, y train)
```

```
# Train SVM model
svm_model = SVM(kernel='linear')  # You can specify different
    kernels like 'linear', 'poly', 'rbf', etc.
svm_model.fit(X_train_tfidf, y_train)
```

With our models trained, we were able to make predictions based off of our test data, which we preprocessed in the same way as we did our training data. We then had to convert our classes back to their original names, from numbers. (Appendix D)

5. Task 2: Experimental Results

The results of our F1 testing reveals big differences in the performance of each model. The logistic regression model achieved F1 results of between 0.05-0.54 for all the 5 categories. The SVM model achieved better results, with F1 results between 0.14 and 0.61. Interestingly, for both models, the "WS" (wit) class achieved the highest F1 score, indicating this was the easiest to classify. In both models, "IR" (irony) and "EX" (exaggeration) achieved lowest scores, indicating difficulty in classifying those types of jokes.

6. Task 3: Experimental Setup

6.1. Data Preparation

The data provided consisted of 3 JSON files: "translation EN FR train input", consisting of 1406 jokes in English, "translation EN FR train qrels", containing 5839 jokes in french, and "task3 2024 test", consisting of 4502 rows of jokes in English. There were also two .tsv files, "joker translation EN-FR train input.tsv, and "joker translation EN-FR train qrels.tsv". In the parent folder, there was also a JSON, "joker translation test", and "joker translation test.tsv".

6.2. Method

First we merged our data to create a unified data frame around the English and French jokes, joined using the "id en" variable. We loaded the hugging face "transformer" library and used the MarianMTModel and MarianTokenizer then used a pre-trained model, "Helsinki-NLP Opus", after trying first with EasyNMT and finding better results from the Helsinki model. (Appendix E)

We iterated through the whole list of English jokes, translating to French, then decoded the vectors back to letters, before saving in a new column alongside the originals. (Appendix F)

7. Task 3: Experimental Results

The translations seemed to be successful however, one drawback of using our method was that the process was highly time intensive, suggesting there may be more practical methods available.

8. Conclusion

We can conclude that our techniques are effective at identifying the presence of a joke, but rather less effective at classifying them into one of our five classes. Thanks to the existence of pre-trained models which are tailored to the task, we can conclude that the translation of our jokes from English to French was successful.

Acknowledgments

We would like to extend our gratitude to the University of Brest for organising the Blended Intensive Programme (BIP) AI For Humanities. We would also like to thank Liana Ermakova for her teaching of the course and Caroline L'haridon for her support during our stay in Brest.

References

[1] C. Jones, B. K. Bergen, People cannot distinguish gpt-4 from a human in a turing test, arXiv, 2024. URL: https://arxiv.org/abs/2405.08007.

[2] A. Baranov, V. Kniazhevsky, P. Braslavski, You told me that joke twice: A systematic investigation of transferability and robustness of humor detection models, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 13701–13715.

[3] F. Góes, P. Sawicki, M. Grześ, D. Brown, M. Volpe, Is gpt-4 good enough to evaluate jokes?, 2024.

[4] J. Hessel, A. Marasovic, J. D. Hwang, L. Lee, J. Da, R. Zellers, R. Mankoff, Y. Choi, Do androids laugh at electric sheep? humor "understanding" benchmarks from the new yorker caption contest, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Toronto, Canada, 2023, pp. 688–714.

[5] S. Jentzsch, K. Kersting, Chatgpt is fun, but it is not funny! humor is still challenging large language models, 2023.

[6] B. M. Savage, H. L. Lujan, R. R. Thipparthi, S. E. DiCarlo, Humour, laughter, learning, and health! a brief review, Advances in Physiology Education (2017).

[7] D. Delabastita, Wordplay as a Translation Problem: A Linguistic Perspective, De Gruyter Mouton, 2004.

[8] A.-G. B. V. M. P. P. G. S. Liana Ermakova, Tristan Miller, A. Jatowt, Overview of clef 2024 joker track on automatic humor analysis, experimental ir meets multilinguality, multimodality, and interaction. proceedings of the fifteenth international conference of the clef association (clef 2024) (2024).

.1. Appendix A

```
results = []
# Iterate over each test query
for index, test query in data test queries.iterrows():
    query id = test query['qid']
    query text = test query['query']
    # Calculate relevance for each joke in the corpus with this
       query
    scores = []
    for  , joke in data corpus.iterrows():
        if joke['text'] is None:
            continue
        else:
          text all = query text + " " + joke['text']
          vectorized text = tfidf vectorizer.transform([text all])
          relevance score = model.predict proba(vectorized text)[0,
             1]
          scores.append({
              'docid': joke['docid'],
              'score': relevance score
          })
```

.2. Appendix B

```
# Sort jokes by relevance score in descending order
scores.sort(key=lambda x: x['score'], reverse=True)
    # Prepare output JSON format
for rank, score info in enumerate(scores, start=1):
    results.append({
        'run id':"Tomislav&Rowan task 1 TFIDF",
        'manual':0,
        'rank': rank,
        'score': score info['score'],
        'docid': score info['docid'],
        'qid': query id
    })


with open('result joker task 1.json', 'w') as outfile:
    json.dump(results, outfile, indent=4)
```

.3. Appendix C

```
# Preprocessing function
from nltk.stem import WordNetLemmatizer
import contractions
import re
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.corpus import stopwords


lem = WordNetLemmatizer()
def preprocess text(text):
      sms = contractions.fix(str(text)) # converting shortened words
          to original (Eg:"I'm" to "I am")
      sms = sms.lower() # lower casing the message
      sms = re.sub(r'https?://S+|www.S+', "", sms).strip() #removing
          url
      sms = re.sub("[^a-z ]", "", sms) # removing symbols and
         numbers (keeping only charachters from a-z)
      sms = sms.split() #splitting
      # lemmatization and stopword removal
      sms = [lem.lemmatize(word) for word in sms if not word in set(
         stopwords.words("english"))]
      sms = " ".join(sms)
      return sms
X = df train["text"].apply(preprocess text)
```

.4. Appendix D

```
df bayes test = pd.DataFrame(test data)
# Apply text preprocessing
df bayes test['clean text'] = df bayes test['text'].apply(preprocess
    text)


# TF-IDF Vectorization for test data
X test tfidf = tfidf vectorizer.transform(df bayes test['clean text
    '])


# Predict
bayes predictions = naive bayes model.predict(X test tfidf)


# Convert back to original names
bayes predicted classes = label encoder.inverse transform(bayes
    predictions)
```

.5. Appendix E

```
from transformers import MarianMTModel, MarianTokenizer


# Load pre-trained MarianMT model and tokenizer for English to
    French translation
model name = "Helsinki-NLP/opus-mt-en-fr"
model = MarianMTModel.from pretrained(model name)
tokenizer = MarianTokenizer.from pretrained(model name)


# Define input text
input text = "Translate this text to French."


# Tokenize input text
inputs = tokenizer(input text, return tensors="pt")


# Perform translation
outputs = model.generate(**inputs)


# Decode translated output
translated text = tokenizer.decode(outputs[0], skip special tokens=
    True)


# Print translated text
print("Translated text:", translated text)
```

.6. Appendix F

```
# Assuming you have already loaded the test data into a DataFrame df
    test data


results = []


# Translate jokes
for  , row in df test data.iterrows():
    # Translate each row's English text to French
    translation = model.generate(**tokenizer(row['text en'], return
       tensors="pt", padding=True))
    translated text = tokenizer.decode(translation[0], skip special
       tokens=True)


    # Append the translation result to the results list
    results.append({
        'run id': "Tomislav&Rowan task 3 MarianMTModel",
        'manual': 0,
        'id en': row['id en'],
        'text fr': translated text
    })


# Convert results list to DataFrame
translated df = pd.DataFrame(results)


# Print or use the translated DataFrame as needed
print(translated df)
```