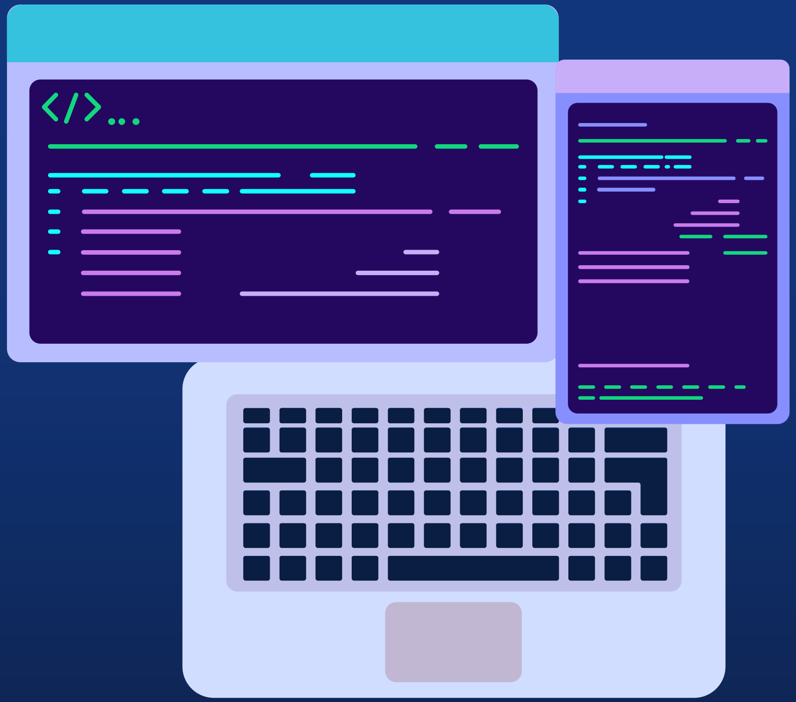
CLEF JOKER Tasks 1-3



Rowan Mann, Christian-Albrechts-Universität zu Kiel (CAU)
Tomislav Mikulandrić, University of Split, Faculty of Science

Overview

Use TF-IDF to rank the jokes for some queries
Try different types of models for joke classification
Compare translation models

Task 1: Humour-aware information retrieval

	qid	docid	qrel	text	query
0	qid_train_0	27260	0	Testament is a written document that outlines ...	testament
1	qid_train_0	28561	1	She was only an Attorney's daughter, but what ...	testament
2	qid_train_0	51135	1	I've inherited a fortune, said Tom, willfully	testament
3	qid_train_0	17068	1	The Hong Kong businessman left a huge estate w...	testament
4	qid_train_0	591	1	My name is Will, I'm a lawyer.	testament
...	...	...	...	...	...
2384	qid_train_8	4030	0	Buds may be specialized to develop flowers or ...	buzz
2385	qid_train_8	58185	1	Waiter, there's a fly in my soup!" I know. It...	buzz
2386	qid_train_8	53481	1	OLD PILOTS never die, they just buzz off.	buzz
2387	qid_train_8	53166	1	Get the buzz on your favorite bee dependent fl...	buzz
2388	qid_train_8	56305	0	The term bud is also used in zoology, where it...	buzz

- Retrieve text relevant to the query, which is also an instance of wordplay
- Combine the data from multiple JSON files
- TF-IDF vectorizer, document is all texts
- Rank the texts for a single query and keep only the important ones
- Documents having the query term rank on top

```
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
          })
```

Task 1: Humour-aware information retrieval

- Created a LogisticRegression model
- Good for binary classification
- Another approach would be to ask an LLM
- Time exhaustive for a lot of texts
- At the end sort the joker by relevance

Task 2: Humour classification according to genre and technique

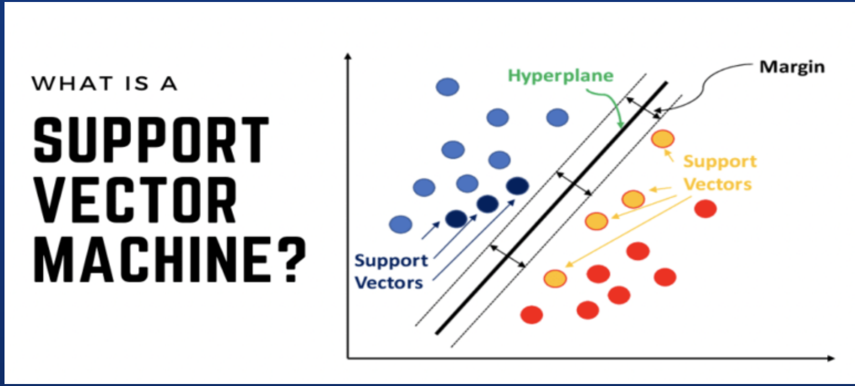
- Load data and merge it with qrels JSON so that we get a training dataframe with classes for each joke
- IR – Irony
- SC – Sarcasm
- EX - Exaggeration
- AID – Incongruity
- SD – Self-deprecating
- WS - Wit
- Preprocess the text and load it to clean_text column

```python
# 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)
```

Task 2: Humour classification according to genre and technique

- Split to train/test datasets
- Train the models and make predicitons
- LogisticRegression – can also be used for multiclass classification
- NaiveBayes – performed the worst
- Support Vector Classifier (SVC)



	precision	recall	f1-score	support
AID	0.65	0.32	0.43	47
EX	0.50	0.08	0.14	38
IR	0.30	0.20	0.24	41
SC	0.46	0.46	0.46	79
SD	0.80	0.25	0.38	32
WS	0.47	0.86	0.61	112
accuracy			0.48	349
macro avg	0.53	0.36	0.37	349
weighted avg	0.50	0.48	0.43	349

Task 3: Translation of puns from English to French

- easyNMT - Easy to use, state-of-the-art Neural Machine Translation
- Automatic download of pre-trained machine translation models
- Can translate between 150+ languages

```
from easynmt import EasyNMT
model = EasyNMT('opus-mt')

#Translate a single sentence to German
print(model.translate('This is a sentence we want to translate to German', target_lang='de'))

#Translate several sentences to German
sentences = ['You can define a list with sentences.',
             'All sentences are translated to your target language.',
             'Note, you could also mix the languages of the sentences.']
print(model.translate(sentences, target_lang='de'))
```

Task 3: Translation of puns from English to French

- MarianMT - A framework for translation models, using the same models as BART
- Transformer style architecture
- Fine-tuned model for English to French translation

```python
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)
```


```json
{
    "run_id": "Tomislav&Rowan_task_3_MarianMTModel",
    "manual": 0,
    "id_en": "en_0",
    "text_fr": "Pour obtenir 20/20 dans un marathon, vous avez vraiment besoin d'avoir r\u00e9vis\u00e9 vos cours."
},
{
    "run_id": "Tomislav&Rowan_task_3_MarianMTModel",
    "manual": 0,
    "id_en": "en_1",
    "text_fr": "Le professeur de maths arrive toujours en portant un nombre impair."
},
```

run_id	count	BLEU	BLEU_1	BLEU_2	BLEU_3	BLEU_4	count	BERT_score_P	BERT_score_R	BERT_score_F1
Arampatzis_GoogleTranslate	376	65,23	78,96	67,48	61,59	57,52	832	91,93%	91,82%	91,85%
Frane_TranslationModel	92	57,13	64,33	58,41	54,66	51,85	279	92,06%	91,53%	91,77%
Dajana&Kathy_TranslationModel	376	58,45	71,94	60,27	54,11	49,73	832	91,35%	91,00%	91,15%
UBO_SDL	312	13,17	71,90	57,17	49,13	43,24	598	90,13%	90,21%	90,15%
Tomislav&Rowan_MarianMTModel	376	58,85	77,11	63,66	56,06	50,45	832	90,82%	89,19%	89,95%
Arampatzis_MarianMT	376	58,85	77,11	63,66	56,06	50,45	832	90,82%	89,19%	89,95%
UBO_ChatGPT	312	13,09	69,90	54,08	46,07	40,31	598	89,12%	89,34%	89,21%
UBO_DeepL	312	11,97	68,53	50,32	41,38	35,11	598	89,06%	89,31%	89,16%
UAms_T5-base_ft	376	48,74	71,75	54,57	45,18	38,05	832	89,53%	88,52%	89,00%
Arampatzis_mBART	376	48,71	70,95	54,40	45,29	38,67	832	88,95%	87,41%	88,13%
Arampatzis_M2M100	376	42,37	68,46	48,73	37,72	29,93	832	88,23%	87,23%	87,70%
UAms_Marian_ft	376	25,69	47,05	28,47	20,74	15,69	832	81,06%	82,53%	81,74%
Tomislav&Rowan_MarianMTModel	1	11,46	100,00	100,00	100,00	100,00	3	84,42%	71,23%	77,26%
Farhan_2	376	14,33	23,68	15,84	12,05	9,32	832	69,38%	77,14%	72,96%
Farhan_1	376	9,21	15,92	9,97	7,65	5,92	832	64,30%	73,18%	68,41%
jokester_MarianMTModel	49	0,29	15,34	0,14	0,08	0,04	112	67,30%	66,38%	66,80%
Arampatzis_opus_mt	63	0,29	15,04	0,23	0,06	0,03	157	66,98%	66,05%	66,47%
Arampatzis_T5	63	0,32	11,35	0,17	0,10	0,06	157	65,91%	64,79%	65,31%

Conclusion

Recent advances in LLM make them a valid choice for humor detection, but LLM's still have issues with grasping humor

Classical classification models should be a better fit for humor type classification
Specialized translation frameworks work better than LLMs

Use of puns may be too subtle for an LLM to acknowledge it

Irony is by far the hardest type of humor for machines to detect

Questions?