

Idioms Extractor

Project by: Mihika Nigam & Kedarnath Chaturvedula

Background

Idioms are expressions or phrases that have a figurative meaning that is different from the literal interpretation of words. These words often carry cultural or contextual significance and are considered an integral part of every language as they are widely used in conversational language as well as an important part of literature. They convey ideas or emotions more colorfully and exaggeratedly which is always interesting to interpret as they add richness to the language and make communication more interesting and engaging.

The role of idioms has always posed a significant problem in the field of natural language processing where the application of extracting sentiments or semantic meanings comes into the picture. This is because the meaning of the entire idiom phrase is often different from the meaning of each phrase. Hence, this also increases the complexity of word sense disambiguation, text summarization, etc where a lot of these phrases are used. Idioms unlike part-of-speech do not have a generalized pattern to them as there is no relation between words in those phrases. The entire phrase as a whole has its separate meaning which is the reason for the complexity to be higher while dealing with idioms.

DataSource & Tools

Our goal is to tag idioms from an input text based on an existing database which has over 9,878 entries of distinct idioms along with their usages and definitions. The idioms were collected from various resources like the Oxford Dictionary and Wiktionary. Capturing all the variations and nuances in the idiom phrases provided in the input text and being able to match that to our dataset is the main crux of this entire goal. There can be multiple variations in idiomatic phrases like tense variations and subtle word changes (like having a preposition before or after) etc.

The tools that we used to develop this project are

- Google Colab
- VSCode(IDE)
- Python3
- NLTK
- Spacy
- JSON

Methodology

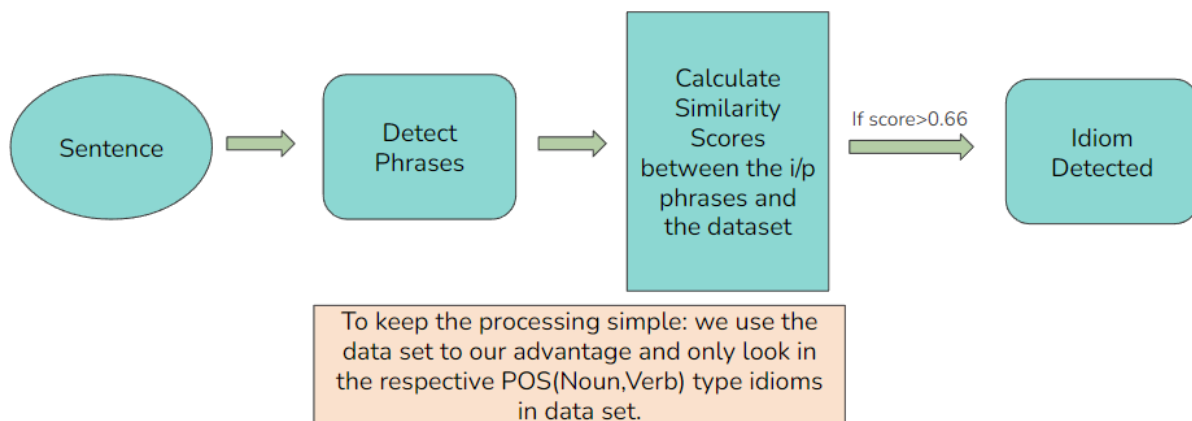
We approached this problem in 2 phases.

Phase1:

Initially, we started with matching each word of the input text with each word of the idiom iterating over all the idioms in the dataset. This was a very time-consuming approach and only handled exact phrases as the idiom variations have not been handled. To handle variations and improvise the functionality, we used the spacy phrase matcher along with POS tagging. Here, we captured the part of speech of every word in the input and handled variations on verbs by converting verbs into their base form using the NTK wordnet lemmatizer. This provided good results and matched exact idioms in the input text to the idiom dataset. It handled variations in tenses and verbs(rain cats and dogs → raining cats and dogs, burning the midnight oil → burning the midnight oil, etc.) However, this strategy also was very time-consuming as it matches each word from the tokenized input to each word in the idiom dataset and iterates over all the idioms. Explaining this in detail:

Phase 2:

Hence, we moved on to the main approach which made the functionality more efficient by extracting phrases from the text. The process flow is as follows:

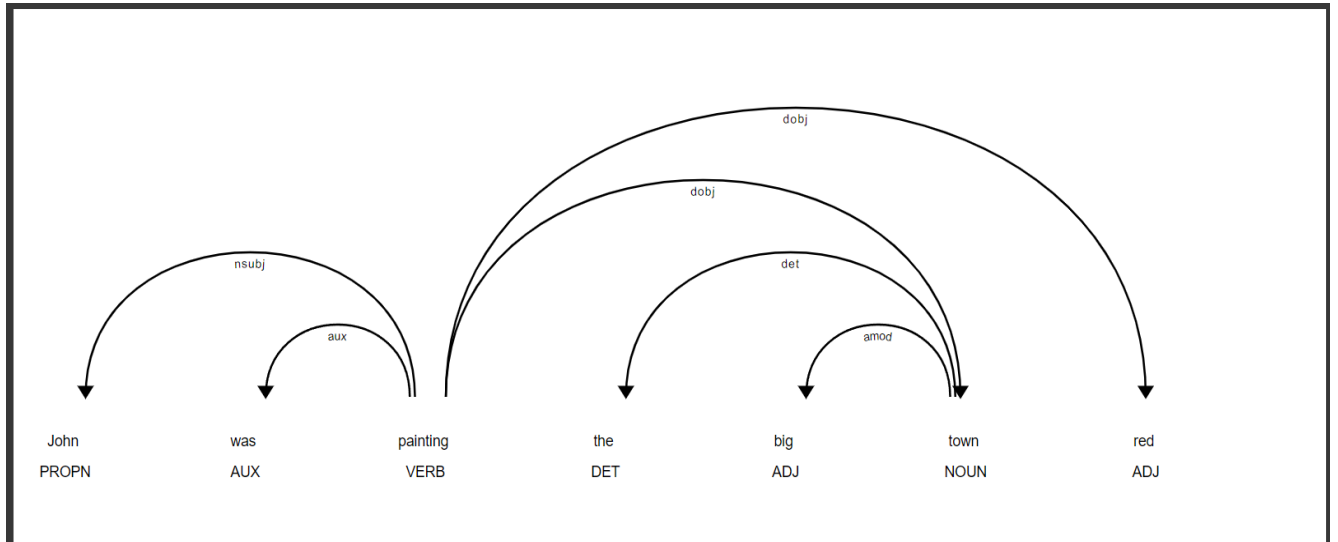


We provide a text paragraph as an input and extract sentences from it. Then, phrases from each of the sentences are detected and extracted from them. Similarity scores are used to calculate the similarity between the incoming phrase and the idiom in the dataset and the phrase is tagged as an idiom if the similarity score is above the threshold 0.66. We arrived upon this value after testing with different values but this provided the most accurate results and also this was the threshold recommended by the SequenceMatcher library documentation from difflib. To keep the processing simple, we only focused on the respective POS(Noun,Verb) type idioms in the dataset.

Explaining in detail:

Step 1: We process the text.

Step 2: After processing the sentences we extract phrases from the sentences. We use spacy to extract noun phrases. For verb phrases we write our own algorithm. We use spacy's visualiser to detect any patterns to detect verb phrases.

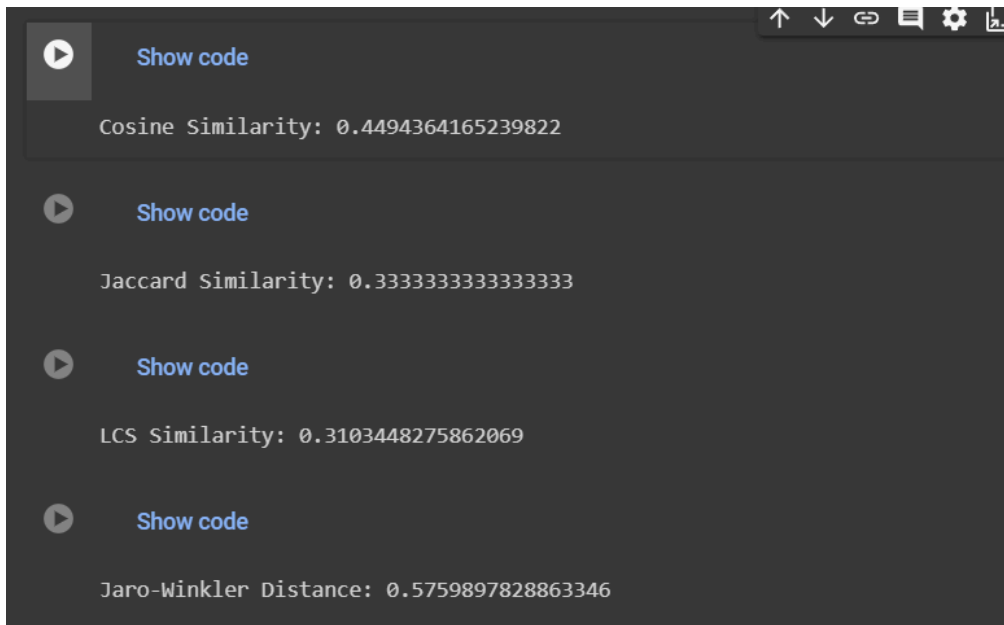


Step 3: We came up with following pattern:

```
def verb_chunks2(doc):
    chunks=set()
    for token in doc:
        if token.pos_ in ['AUX','VERB']:
            subtree=[node for node in token.subtree]
            print(f"TOKEN : {token}, {token.tag_}, SET :", subtree)
            arcwords=[]
            for child in subtree[subtree.index(token):]:
                print(f'{child}, subtree : {child.dep_}')
                arcwords.append(child.text)
                if child.dep_ in (['doobj','attr','aux','prt'] if token.dep_=='ROOT' else ['doobj','attr','prt']):
                    break
            print('my arc words: ',arcwords)
            if token.dep_=='ROOT' or token.dep_=='advcl':
                chunks.add(' '.join([right.text for right in subtree[subtree.index(token):]]))
            chunks.add(' '.join(arcwords))
            print('chunks now: ',chunks)
    print('\n')
    return list() if chunks==set() else list(chunks)

verb_chunks2(doc)
```

Step 4: We pass through each noun and verb phrase we extracted through the idiom matcher. To pick the right similarity score calculator we tried a lot of inbuilt algorithms.



```

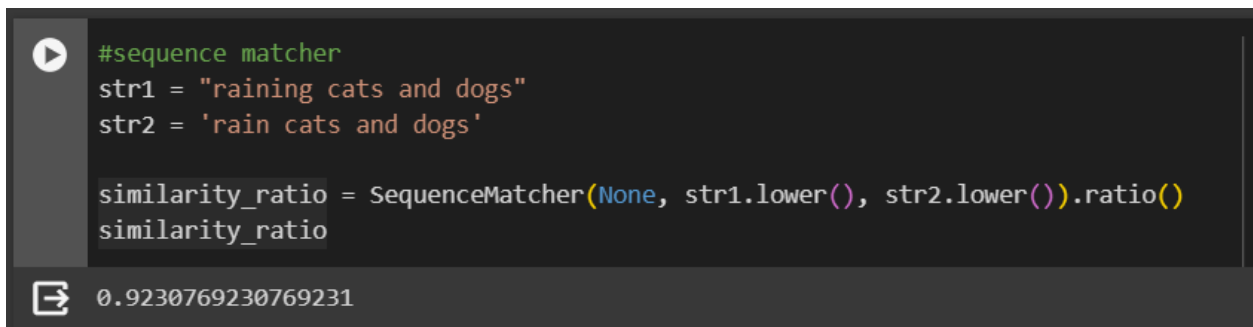
Show code
Cosine Similarity: 0.4494364165239822

Show code
Jaccard Similarity: 0.3333333333333333

Show code
LCS Similarity: 0.3103448275862069

Show code
Jaro-Winkler Distance: 0.5759897828863346
```

We decided to go with diffLab's SequenceMatcher that follows the concept of matching Lowest Common Subsequence.



```

#sequence matcher
str1 = "raining cats and dogs"
str2 = 'rain cats and dogs'

similarity_ratio = SequenceMatcher(None, str1.lower(), str2.lower()).ratio()
similarity_ratio

0.9230769230769231
```

Step 5: To improve accuracy for 2 worded phrases we increased the similarity score threshold (= 0.84):

```
def tag_idioms(phrase,pos, dataset):
    doc1=nlp(phrase)
    maxscore=0.84 if len(phrase.split())<=2 else 0.67; #more accuracy for 2 word phrases
    idiom=None
    for idiom_data in dataset:
        i = idiom_data["idiom"]
        score=SequenceMatcher(None, i.lower(), phrase.lower()).ratio()
        if maxscore<score:
            maxscore=score
            idiom=i
    return idiom
```

Then finally we get our idioms.

Evaluation & Results

For phase 1 as it is a straightforward approach we provided a paragraph with idioms in and outside the dataset and it captured them all with variations with higher execution time.

Results of Phase1:

- It was able to detect all the idioms there in the dataset
- This is a sample input given which was multiple idioms and the phase 1 algorithm detected all idioms which were present in the dataset.

```
input_text="Navigating through the labyrinth of life, Jake decided to bite the bullet and spill the beans about his secret project, letting the cat out of the bag. As he took the bull by the horns, he realized that he was walking on eggshells, hoping not to stir the hornet's nest. Despite facing an uphill battle, he knew that every cloud has a silver lining and that revealing the ace up his sleeve was a piece of cake. As he awaited the reaction, he kept his fingers crossed, knowing that the ball was in their court, and it was time to play his cards right."
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + v [ ] [ ] ... ^
PS C:\Users\BWIN\Desktop\IdiomExtractor> python idiom_extractor.py input.txt
Enter 1 or 2 below
: Option 1: A fast and efficient approach for tagging idioms ,
: Option 2: A Direct idiom matching based on pattern matching by token variations which takes longer time.
2
[{'idiom': 'bite the bullet'}, {'idiom': 'spill the beans'}, {'idiom': 'let the cat out of the bag'}, {'idiom': 'take the bull by the horns'}, {'idiom': 'walk on eggshells'}, {'idiom': 'uphill battle'}, {'idiom': 'a silver lining'}, {'idiom': 'piece of cake'}]
PS C:\Users\BWIN\Desktop\IdiomExtractor>
```

Results of Phase2:

- It was able to detect all the phrases which were rightly associated with the database under the correct part of speech.
- For phase 2 we evaluated the examples with 45 different sets and the results are as follows.

Here are the other output samples which we tested and evaluated against.

```
test_input = [
    {"when i came home, he let me have it for wrecking the car": ["let someone have it"]},
    {"She walked to the store to buy some groceries.": []},
    {"John and Mary painted the town red.": ["paint the town red"]},
    {"The quick brown fox jumps over the lazy dog.": []},
    {"He spilled the beans about the surprise party.": ["spill the beans"]},
    {"The team hit the ground running after the war talk.": []}
]
```

```
Expected Output : ['let someone have it']
My phrases :
{'noun': ['i', 'it', 'the car', 'me', 'he'], 'verb': ['let me have it', 'wrecking the car', 'let me have it for wrecking the car', 'have it', 'came home']}
Our Output : {'let them have it', 'have at'}

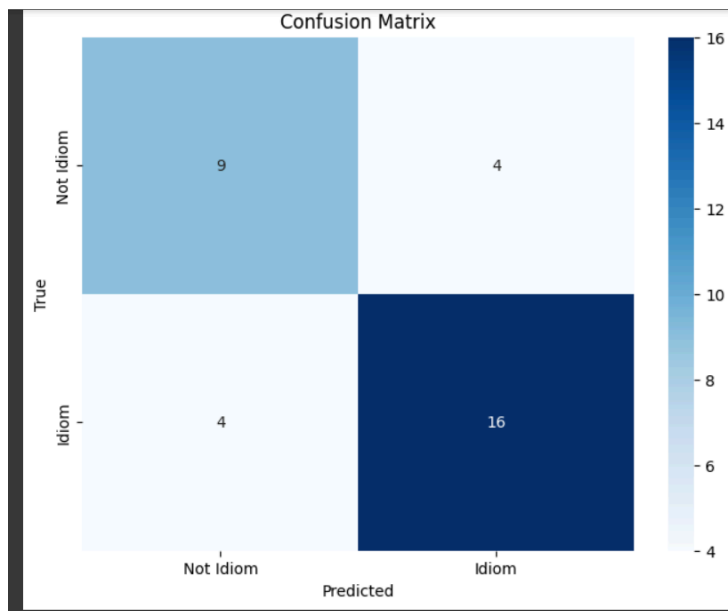
Expected Output : []
My phrases :
{'noun': ['the store', 'she', 'some groceries'], 'verb': ['walked to the store to buy some groceries .', 'buy some groceries', 'walked to the store to']}
Our Output : []

Expected Output : ['paint the town red']
My phrases :
{'noun': ['John', 'Mary', 'the town red'], 'verb': ['painted the town red', 'painted the town red .']}
Our Output : {'paint the town red'}

Expected Output : []
My phrases :
{'noun': ['the lazy dog', 'The quick brown fox'], 'verb': ['jumps over the lazy dog .']}
Our Output : []

Expected Output : ['spill the beans']
My phrases :
{'noun': ['He', 'the surprise party', 'the beans'], 'verb': ['spilled the beans', 'spilled the beans about the surprise party .']}
Our Output : {'spill the tea'}
```

This is the result analysis after analyzing the output.



The entire output analyzed is present in the google colab.

Explaining shortcomings:

- The rules we wrote for verb phrase detection couldn't cover the right set of phrases. Detecting the base level phrase and then the phrases at the leaf was a tedious task.
- While evaluating similarity scores a phrase was passed as an idiom if it passed the minimum threshold, so that led to a lot of false positives.

Shortcomings

1. **High False Positive Rate (20%):** The current implementation results in a significant number of false positives, compromising the precision of idiom detection.
2. **Challenges in Similarity Score Algorithm:** Inability to fine-tune the similarity score algorithm contributes to the generation of false positives, highlighting a need for optimization.
3. **Verb Phrase Detection Limitations:** The rules designed for verb phrase detection lack coverage for the appropriate set of phrases, indicating room for improvement in capturing diverse language patterns.
4. **Limited Handling of Grammatical Mistakes:** The detector struggles to handle grammatical mistakes, revealing a weakness in error tolerance that could be addressed for enhanced robustness.

5. **Confusion with Long and Complex Sentences:** The detector exhibits confusion when faced with very long and complex sentences, indicating a need for better handling of intricate linguistic structures.

Future Work

Possible Future Work for our NLP Project Idioms Detector include the following major points:

1. **Expansion to Prepositions and Adjectives:** Extend idioms detector to adjectives and preposition detector.
2. **Fine-Tuning and Optimization:** Invest efforts in fine-tuning and optimizing the codebase to improve the overall accuracy of results. This involves refining algorithms, adjusting parameters, and implementing efficiency measures to ensure more reliable and precise idiom detection.
3. **Dynamic Threshold Adjustment:** Implement a dynamic threshold adjustment mechanism for similarity scores to adapt to varying linguistic contexts. This can help mitigate false positives and enhance the detector's adaptability to different writing styles and genres.
4. **Machine Learning Integration:** Explore the integration of machine learning techniques to allow the detector to learn and adapt from data patterns. This approach can contribute to the continuous improvement of the model's performance as it encounters new linguistic nuances.
5. **Error Correction Mechanism:** Develop an error correction mechanism to address grammatical mistakes, enabling the detector to handle imperfect language structures more effectively and providing users with more accurate results.

Conclusion

The "Idioms Extractor" project focuses on the challenge of tagging idioms in text using a database of over 9,878 idioms. It employs tools like Google Colab, Python3, and Spacy. The methodology involves two phases: the first uses exact matching of idioms, while the second uses phrase extraction and similarity scoring. The project achieved accurate idiom detection but faced limitations like high false positive rates and challenges in verb phrase detection and handling complex sentences. Future work includes expanding phrase detection and refining the code for better accuracy. The project highlights both the complexity and potential of natural language processing in idiom detection.

Works Cited

Penn Treebank P.O.S. Tags,

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html. Accessed 14 December 2023.

Grey, Alexander. "The complete guide to string similarity algorithms | by Yassine EL KHAL | Medium."

Yassine EL KHAL, 21 August 2023,

<https://yassineelkhal.medium.com/the-complete-guide-to-string-similarity-algorithms-1290ad07c6b7>. Accessed 14 December 2023.

"ICE: Idiom and Collocation Extractor for Research and Education." *ACL Anthology*, 3 April 2017,

<https://aclanthology.org/E17-3027.pdf>. Accessed 14 December 2023.

"A New Approach for Idiom Identification Using Meanings and the Web." *Department of Computer*

Science, <https://www2.cs.uh.edu/~rmverma/ranlp.pdf>. Accessed 14 December 2023.