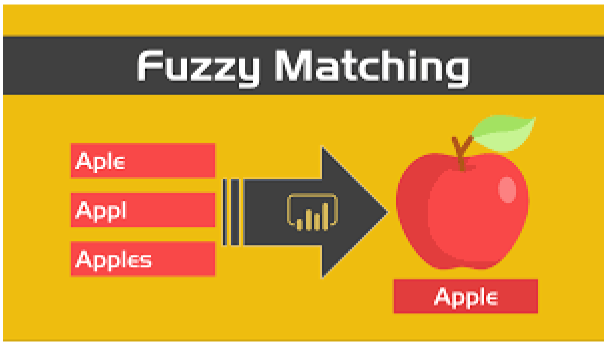
Fuzzy Matching Assignment 1 Suvanth Ramruthen

What is Fuzzy Matching

Fuzzy matching is a technique used to compare and match two strings that are not exactly identical. While the naive approach of comparing strings and checking for equality would only make a direct comparison that is binary and doesn't consider the degree to which two strings might match each other. Fuzzy matching algorithms use various techniques to measure the similarity between two strings, even if they're not exactly identical. These techniques can include things like measuring the number of characters or words that are the same, or looking for common patterns or sequences of characters. By taking a more nuanced approach to comparing strings, fuzzy matching can help identify matches even when there are slight differences or variations in the data being compared.



In [ ]:

```
from thefuzz import fuzz, process # python lib for fuzzy logic

string1 = "Hello World"
string2 = "hello world"

print(string1 == string2)  # True or False equality
print(f'The strings have a {fuzz.partial_ratio(string1,string2)}% similarity
```

```
False
The strings have a 82% similarity
```

The codeblock above shows the value of using Fuzzy Matching. We could have potentially lost out on a word/phrase that has a signifcantly high similarity.

Use cases of Fuzzy Matching

- Clean Customer Data: Companies use fuzzy matching to find and merge duplicate customer records. This helps to keep the customer database accurate and up-to-date, which is important for sales and marketing efforts.

- Match Medical Records: Healthcare providers use fuzzy matching to match patient records across different hospitals or clinics. This helps to ensure that patients receive consistent and accurate medical care.
- Detect Fraud: Fuzzy matching can help to identify suspicious patterns in financial data, such as transactions that are similar but not identical. This can be used to detect potential fraud.
- Personalize Product Recommendations: E-commerce websites use fuzzy matching to recommend products to users based on their search history or past purchases. This can help to improve the user experience and increase sales.
- Verify Addresses: Postal services and shipping companies use fuzzy matching to verify and correct address information. This helps to ensure that packages are delivered to the correct address and reduces the risk of misdeliveries or lost packages.

Popular Software that leverage Fuzzy Matching

Elasticsearch is a search and analytics engine that can use fuzzy matching to improve the accuracy of search results. It can match queries with documents that have similar terms or phrases, even if they are spelled differently or have different word forms.

Apache Lucene is a full-text search engine library that supports fuzzy matching to help users find documents that contain similar terms or phrases. It can also use fuzzy matching in conjunction with other search features, such as stemming and synonyms.

Redis is an in-memory data structure store that can be used as a database, cache, and message broker. It includes a fuzzy matching module called Redisearch that can be used to perform full-text search and fuzzy matching queries on data stored in Redis. Redisearch uses an inverted index to index and search documents, which allows it to perform fuzzy matching queries efficiently.

Approaches to computing similarity of two Strings

Lets consider two strings String 1 and String 2

- Number of edits required to transition from String 1 to String 2
  - Edits would consist of the number of character replacements, deletions or insertions.
- Common word counts/tallies
- Frequency of letters and phrases
- Longest common Substrings

Overview of core Fuzzy Matching Algorithms

Levenshtein distance

The Levenshtein distance algorithm is a way to compare two words or strings to see how different they are from each other. It counts how many changes you need to make to one word to turn it into the other. The changes can be deleting a letter, adding a letter, or changing a letter. The fewer changes you need to make, the more similar the words are.

Soundex and Metaphone

Soundex is an algorithm that is used to index words by their sound, so that similar-sounding words can be matched even if they are spelled differently. It works by taking a word, dropping its vowels (except for the first letter), and assigning a code to each of the remaining consonants based on their sound. Consonants that have the same sound get the same code. The resulting code can then be used to compare words and find matches.

Metaphone is an algorithm that indexes words by their pronunciation, just like Soundex. It's better than Soundex because it can handle more complex phonetic patterns and a wider range of words from different languages. It works by assigning codes to the sounds in a word based on their phonetic properties and the context of the sounds. The resulting codes are shorter and easier to read than Soundex codes.

In [ ]:

```python
import jellyfish

name1= "Phillip"
name2= "Fillipe"

soundexCode1 = jellyfish.soundex(name1)
soundexCode2 = jellyfish.soundex(name2)
metCode1 = jellyfish.metaphone(name1)
metCode2 = jellyfish.metaphone(name2)
print(soundexCode1)
print(soundexCode2)
print(metCode1)
print(metCode2)
print(f'{fuzz.partial_ratio(name1, name2)}% similarity')
```

```
P410
F410
FLP
FLP
71% similarity
```

The above codeblock shows that the Soundex algorithm produces similar codes showing similar sounding words, but metaphone shows that they sound the exact same. Levenshtein distance doesnt produce a satifatory result with just 71% similarity.

Cosine Similarity

For cosine similarity between two terms we make use of the following identity

cos(θ) = X · Y / |X| . |Y|

We can obtain the vector of our words using the Bag of words encoding technique.

We have two key values to consider cos(90) and cos(0). If the value of thetha is 90 that results in our identity producing 0 showing no similarity. if the value of theta is 0 that results in our identity producing a value of 1. The result shows a percentage simialrity between the words.

Fuzzy matching can have some problems, including:

- Ambiguity: Fuzzy matching can produce multiple possible matches, leading to uncertainty about which match is the most relevant and similar
- Performance: Some algorithms can be slow and struggle with large amounts of data. This necessitates the use of advanced techniques such as Dynamic programming.
- Sensitivity to input data: Fuzzy matching can be affected by errors in the input data, such as misspellings or incorrect capitalization.
- Parameter tuning: Some fuzzy matching algorithms have parameters that need to be adjusted to work optimally. Tuning is an iterative process that is often diccult to interpret performance increases/changes

In [ ]:

```
#Matching from a array
arrFuzzTest = ["Programming", "Kotlin Language", "Java Language", "Spark Fra
print(process.extract("Language", arrFuzzTest, limit=3))
```

```
[('Kotlin Language', 90), ('Java Language', 90), ('English language', 90)]
```

The above codeblock displays illogical results, displays the need for more advance techniques such as NLP

Python Pandas Fuzzy Matching Demonstration

In [ ]:

```
import pandas as pd
import numpy as np


# creating the dictionaries
arr1 = ["dog", "berry", "choco", "peanut", "aptly"]
arr2 = ["dog canine", "blueberry", "raspberry","doggies", "doggo", "peanuts"

dframe1 = pd.DataFrame(arr1, columns=['Items'])
dframe2 = pd.DataFrame(arr2, columns=['Items'])
dframe1
```

Out[ ]:

	Items
0	dog
1	berry
2	choco
3	peanut
4	aptly

In [ ]:

```
# empty lists for storing the matches later
matches1 = []
# matches2 = []

list1 = dframe1['Items'].tolist()
list2 = dframe2['Items'].tolist()

# iterating through list1 to extract
# it's closest match from list2
for i in list1:
    matches1.append(process.extract(i, list2, scorer=fuzz.partial_token_sort
dframe1['matches'] = matches1

print("\nDataFrame after Fuzzy matching:")
dframe1
```

```
DataFrame after Fuzzy matching:
```

Out[ ]:

	Items	matches
0	dog	[(dog canine, 100), (doggies, 100)]
1	berry	[(blueberry, 100), (raspberry, 100)]
2	choco	[(doggo, 40), (dog canine, 20)]
3	peanut	[(peanuts, 100), (raspberry, 33)]
4	aptly	[(apply, 80), (raspberry, 40)]

Conclusion

In conclusion, the above research and code blocks support the findings that fuzzy matching is a valuable tool for comparing words or phrases, as it allows us to consider potential matches that we may not have otherwise examined. While Fuzzy logic represents a significant extension of Boolean logic for measuring similarity, the analysis reveals that it can still return some results that are not entirely sensible. Nonetheless, fuzzy matching remains more effective than traditional equality checks for comparing text strings. The exercise further highlights the importance of optimizing fuzzy search algorithms by utilizing different techniques and identifying the most suitable algorithm for the specific use case. To achieve higher accuracy and relevance of matches, it is necessary to leverage more advanced natural language processing (NLP) and information retrieval techniques.

References

- https://towardsdatascience.com/fuzzy-string-matching-in-python-68f240d910fe
- https://www.analyticsvidhya.com/blog/2021/07/fuzzy-string-matching-a-hands-on-guide/?
- https://www.youtube.com/watch?v=1jNNde4k9Ng
- https://www.youtube.com/watch?v=SoZ1CVU2DdE
- https://nanonets.com/blog/fuzzy-matching-fuzzy-logic/
- https://www.sciencedirect.com/topics/computer-science/cosine-similarity#:~:text=Cosine%20similarity%20measures%20the%20similarity,document%2
- https://medium.com/@ievgenii.shulitskyi/phonetic-matching-algorithms-50165e684526

Libraries

- Pandas
- Numpy
- thefuzz
- jellyfish
- python-levenshtein
- python-levenshtein-wheels

Install using: `pip install <library_name>`

Sample code using thefuzz Library

In [ ]:

```
from thefuzz import fuzz, process

'''
Fuzzy string matching is similar to using regex operation, equal operands ar
'''

string1 = "Hello World"
string2 = "hello world"
string3 = "hello world !"
print(string1 == string2) ## checking for exact equality amongst strings
#Could potentially convert strings to a standard form and compare
print(string1.lower() == string2) ## checking for exact equality amongst str
print(string1.lower() == string3) ## strings are similar but due to no exact


string4 = "Just a thefuz experiment"
string5 = "just a thefuz experiment"
print(f'The similarity between both the sentences using fuzzy logic is {fuzz


string6 = "Just a thefuz experiment"
string7 = "just a thefuz experiment and some code"
print(f'The similarity between both the sentences using fuzzy logic is {fuzz
print(f'The similarity between both the sentences using fuzzy logic is {fuzz
# partial ratio looks at substrings within our string

string8 = "The quick brown fox jumped of the lazy hen"
```

```
string9 = "The lazy hen was jumped over by the quick brown fox"
print(f'The similarity between both the sentences using fuzzy logic is {fuzz
print(f'The similarity between both the sentences using fuzzy logic is {fuzz
print(f'The similarity between both the sentences using fuzzy logic is {fuzz
```

```
False
True
False
The similarity between both the sentences using fuzzy logic is 96%
The similarity between both the sentences using fuzzy logic is 74%
The similarity between both the sentences using fuzzy logic is 96%
The similarity between both the sentences using fuzzy logic is 49%
The similarity between both the sentences using fuzzy logic is 52%
The similarity between both the sentences using fuzzy logic is 88%
```

Github Link

https://github.com/Suvanth/Fuzzy-Matching