Northeastern University

Khoury College of Computer Sciences

CS 5800 Algorithms

Final Project

Spring 2022

Akhila Sulgante

Kasi Viswanath Vandanapu

Shital Waters

Under the guidance of
Professor Lama Hamandi

Contents

Introduction

This project is an experiential project in collaboration with the Cordiance Company. The goal of this project is to **get the closest possible** UNSPSC code match for the Avalara Tax description. To begin this project, we were provided with two data files: UNSPSC and Avalara.

UNSPSC file contains code to be matched and it is divided into 4 levels:

- Commodity level
- Class level
- Family level
- Segment level

Avalara file contains Avalara Tax System code, its description and additional information related to the same.

The suggested methodology to map the code was String matching using the Levenshtein distance algorithm. It was also asked to look for Data Truncations and spelling errors to get better matches.

Data Insights

UNSPSC file contains the following product information at each level

- Level Code: This is the UNSPSC code to be matched with Avalara text
- Level Title: This is the string that is to be compared with Avalara text to get the UNSPSC code
- Level Definition: This contains a detailed explanation of products that are included at a particular level
- Synonym: Translation of commodity level title in various languages such as Japanese, Chinese, Korean, French, German etc.
- Acronym: Short forms of few of the Commodity level titles

Unique values at each level:

Level	**No. Of Unique values**
Segment Title	57
Family Title	546
Class Title	7902
Commodity Title	147893

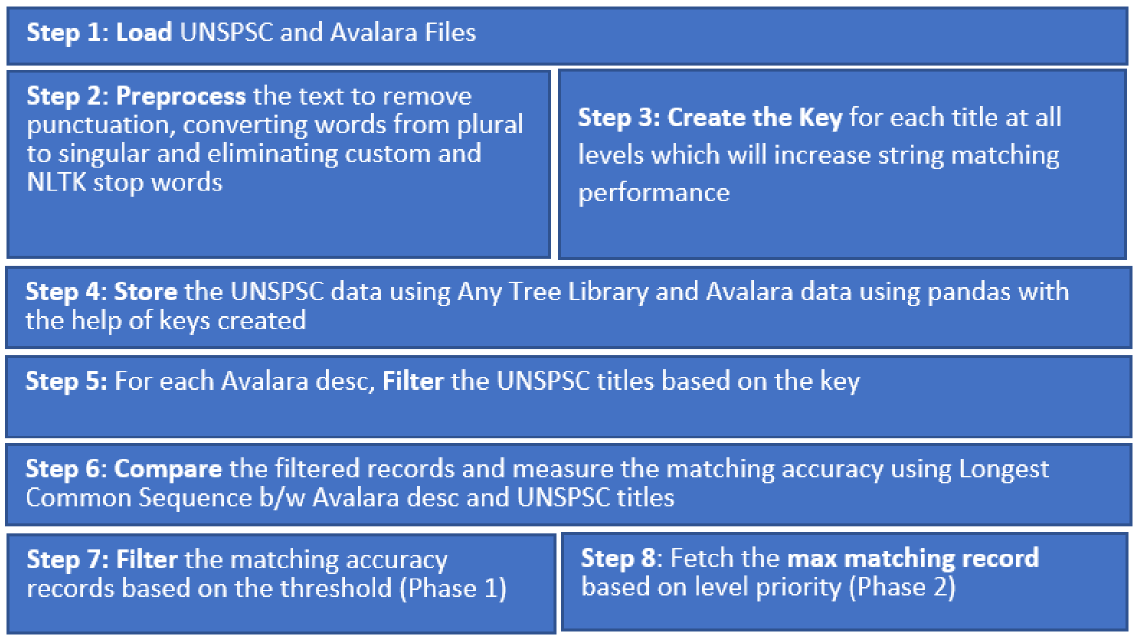
Avalara file contains the following information

- Avalara Tax code: Tax code of the Avalara system
- Avalara Tax Description: Description of the Tax code
- Additional Information: Additional information on the Description of the Tax code

Lines to be considered for our group were from line 1706 to the end of the file, a total of 840 records.

Methodology

Flowchart of the approach



Step 1: Load UNSPSC and Avalara Files

Step 2: Preprocess the text to remove punctuation, converting words from plural to singular and eliminating custom and NLTK stop words

Step 3: Create the Key for each title at all levels which will increase string matching performance

Step 4: Store the UNSPSC data using Any Tree Library and Avalara data using pandas with the help of keys created

Step 5: For each Avalara desc, **Filter** the UNSPSC titles based on the key

Step 6: Compare the filtered records and measure the matching accuracy using Longest Common Sequence b/w Avalara desc and UNSPSC titles

Step 7: Filter the matching accuracy records based on the threshold (Phase 1)

Step 8: Fetch the **max matching record** based on level priority (Phase 2)

We produced an algorithm that is a combination of String matching and uses a few Natural Language Processing Libraries. Since the provided data was not in normalized fashion there was no scope to get accurate matches by using just String-matching algorithms.

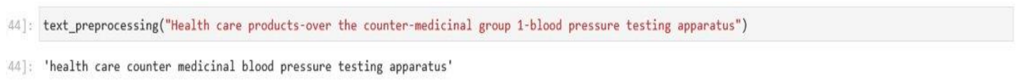
For instance, if either the Levenshtein distance algorithm and longest or common sub-sequence algorithm were used, the probability of getting meaningful matches was 3~5%.

Our Algorithm builds on the principle of Longest common sub-sequence but with little alterations.

Instead of comparing strings from Avalara and UNSPSC files directly, we first subject the Avalara strings to text preprocessing in which the strings are checked for singular and plural form by inflect engine[1] library, if the String has plural words, then it is converted to singular, remove all punctuation, numbers and eliminate all Stop words. Stop words are general words of any language, apart from the pre-defined stop words from NLTK[2] library we analyzed the Avalara strings and came up with custom stop words.

Below is an example of text preprocessing:

Fig: 3.1

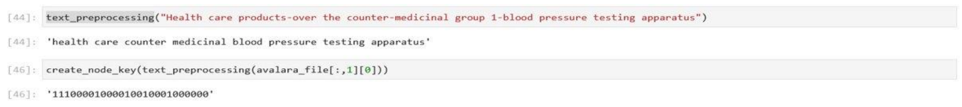


```
44]: text_preprocessing("Health care products-over the counter-medicinal group 1-blood pressure testing apparatus")

44]: 'health care counter medicinal blood pressure testing apparatus'
```

Once the Avalara string is preprocessed, instead of comparing the preprocessed string with UNSPSC strings, we generate a 26-bit array where the string is broken down into words and the first character of each string is extracted and the value at that corresponding alphabet index of the array is stored as 1 and if a character is not present as the first character of words in a string, then the value at that index is stored as 0.

Below is an example of a 26-bit generated array

Fig: 3.2

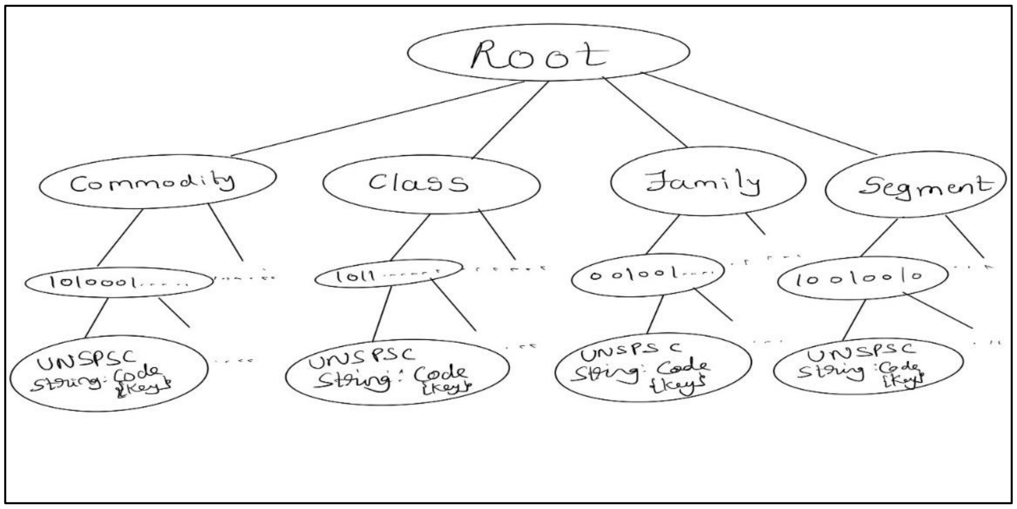


```
[44]: text_preprocessing("Health care products-over the counter-medicinal group 1-blood pressure testing apparatus")

[44]: 'health care counter medicinal blood pressure testing apparatus'

[46]: create_node_key(text_preprocessing(avalara_file[:,1][0]))

[46]: '11100001000010010001000000'
```

This array is generated for every string of Avalara and UNSPSC file and the corresponding 26-bit array is stored as a key.

The data structure used to store the UNSPSC data is tree and that of Avalara is Array.

Fig: 3.3

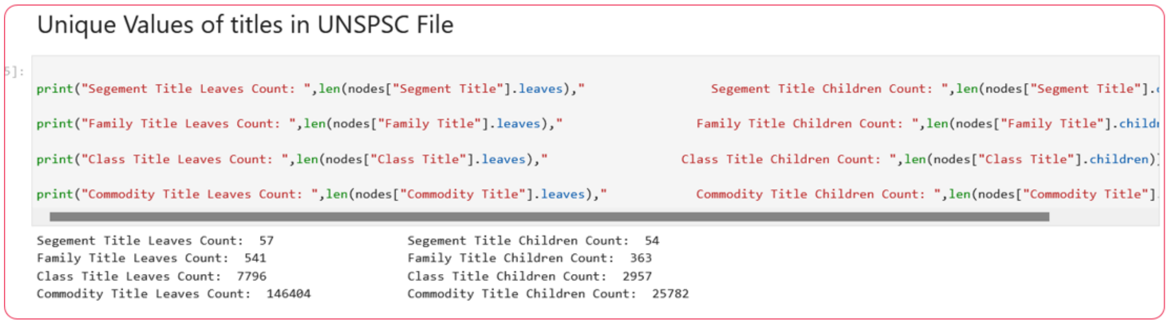

Root
Commodity
Class
Family
Segment
101000l.....
1011.......
00100l.....
10010010
UNSPSC String: Code {Key}
UNSPSC String: Code {Key}
UNSPSC String: Code {Key}
UNSPSC String: Code {Key}

The UNSPSC tree is illustrated in the above diagram. Avalara array is stored as below diagram.

By storing the data in the tree (fig 3.3) we are reducing the comparisons to be made while retrieving the UNSCPSC code that in turn improves the efficiency of the algorithm.

Below image shows the reduced node count obtained by storing data using generated keys

Fig. 3.4



Unique Values of titles in UNSPSC File

```
print("Segement Title Leaves Count: ",len(nodes["Segment Title"].leaves),"          Segement Title Children Count: ",len(nodes["Segment Title"].

print("Family Title Leaves Count: ",len(nodes["Family Title"].leaves),"          Family Title Children Count: ",len(nodes["Family Title"].childr

print("Class Title Leaves Count: ",len(nodes["Class Title"].leaves),"          Class Title Children Count: ",len(nodes["Class Title"].children)

print("Commodity Title Leaves Count: ",len(nodes["Commodity Title"].leaves),"          Commodity Title Children Count: ",len(nodes["Commodity Title"].
```

```
Segement Title Leaves Count:  57              Segement Title Children Count:  54
Family Title Leaves Count:  541               Family Title Children Count:  363
Class Title Leaves Count:  7796               Class Title Children Count:  2957
Commodity Title Leaves Count:  146404         Commodity Title Children Count:  25782
```

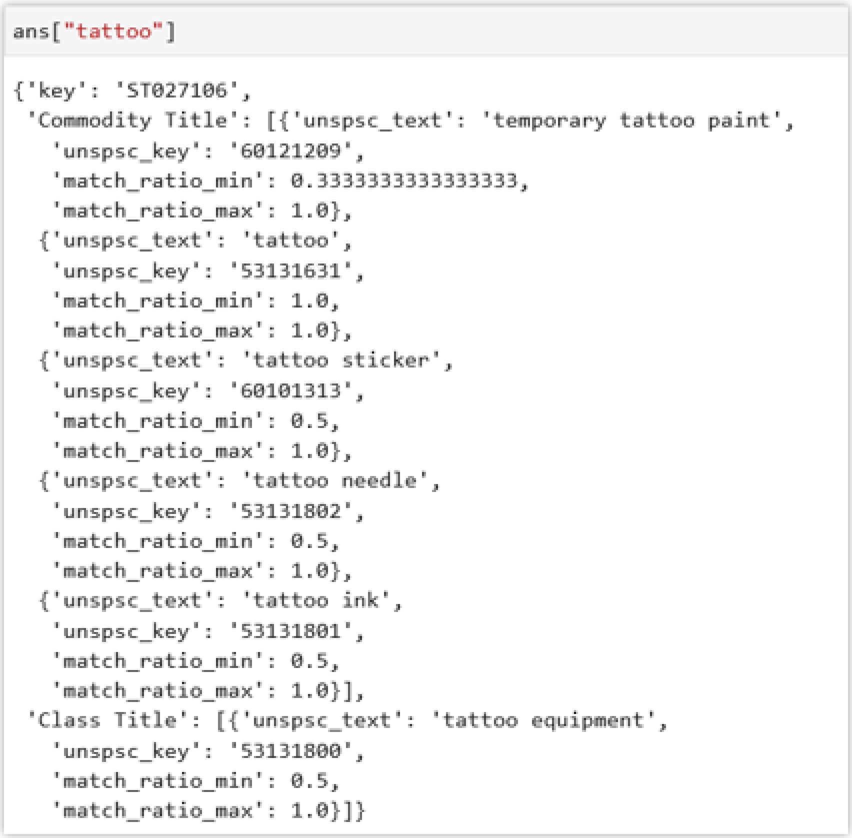
Once the data structure of both the files is ready, we compare the 26-bit generated key array and filter the unnecessary comparison when performing string comparisons.

To illustrate:

String “Alpaca” will have the key as “1000000000000000000000000000”

String “Over-the-counter medicinal products - foot pads, insoles” will have the key as ”0010010010001001000000000000”

Fig. 3.5



```
ans["tattoo"]
```

```
{'key': 'ST027106',
 'Commodity Title': [{'unspsc_text': 'temporary tattoo paint',
   'unspsc_key': '60121209',
   'match_ratio_min': 0.3333333333333333,
   'match_ratio_max': 1.0},
  {'unspsc_text': 'tattoo',
   'unspsc_key': '53131631',
   'match_ratio_min': 1.0,
   'match_ratio_max': 1.0},
  {'unspsc_text': 'tattoo sticker',
   'unspsc_key': '60101313',
   'match_ratio_min': 0.5,
   'match_ratio_max': 1.0},
  {'unspsc_text': 'tattoo needle',
   'unspsc_key': '53131802',
   'match_ratio_min': 0.5,
   'match_ratio_max': 1.0},
  {'unspsc_text': 'tattoo ink',
   'unspsc_key': '53131801',
   'match_ratio_min': 0.5,
   'match_ratio_max': 1.0}],
 'Class Title': [{'unspsc_text': 'tattoo equipment',
   'unspsc_key': '53131800',
   'match_ratio_min': 0.5,
   'match_ratio_max': 1.0}]}
```

The above string will not be compared as the keys do not match at all.

After the key comparisons, the corresponding strings are compared with Longest common subsequence algorithm. The algorithm works on WORDS for Longest common subsequence rather than characters.

Once the titles between UNSPSC and Avalara are compared, longest common subsequence minimum match and maximum match by the following formulas, the records are filtered based on the threshold. In this case we were considering threshold as 50%

lcs = longestCommonSubsequence(unspsc_desc, avalara_desc)

match_perc_min = lcs/max_length(unspsc_desc, avalara_desc)

match_perc_max = lcs/min_length(unspsc_desc, avalara_desc)

We use minimum match percentage because if the string is the substring of the other, we will get the match ratio as 100% and we get a lot of possibilities. To get the best match we started using the above formulas which can be observed in fig 3.5

To get the best most accurate match, we follow the following steps:

1) compute 1-x["match_ratio_min"]) *(x["match_ratio_min"]-x["match_ratio_max"]

2) Sort the computed value from step 1 (ascending order) in each level

3) If there are multiple possibilities in each level pick the max(match_ratio_min)

4) For all levels take max(match_ratio_min) with level priority

Specifications

Coding language: Python

Data Structure: Anytree[3]

Text editor: jupyter notebook

Iterations

Iteration 1: Without Stop words

In the first attempt, to match codes from the UNSPSC file to Avalara text, the algorithm was working just on the Longest common subsequence on words, and the text was just preprocessed for plural forms and punctuation. The algorithm would match texts that had generic words, such as “Water testing kit” would match with “sand testing test”, here since 2/3 of words are the same the algorithm would output as match found and assign the corresponding UNSPSC code to Avalara text. So, the accuracy was about 40%.

Iteration 2: With Stop Words

In the second attempt, to improve accuracy we came up with stop words. At first, we considered only pre-defined custom words from NLTK library, but the accuracy was still at 48%. So, we decided to manually go through the data and come up with custom stop words. As it is difficult to keep track of the number of times a word appears and its relevance to the data mapping. We created a data visualization chart via Observable[4].

Link for the same:

https://observablehq.com/@kasivisu4/avalara-data-analysis

After adding custom stop words our accuracy for possible matches at all levels shot up to 90% and after extracting one match for each product, we achieved an accuracy of 67%

Testing and Results

Our Algorithm runs at the time complexity of O(mn) and space complexity of O(m+n)

We created a test dataset of 30 records by manually mapping the corresponding UNSPSC code to Avalara description. Our algorithm is divided into two phases

Phase 1(Possible Outcome): In this phase, we were able to get one possibility for each level. Out of 30 we were able to match 27 possibilities

Phase 2: In this phase, we will get the max possibility of all the levels from phase 1. Out of 30 records we were able to match 20 records which gives us the accuracy of **66.7%**

Fig 5.1

Avalara Key	Avalara Desc	UNSPSC Level	UNSPSC KEY	UNSPSC TITLE	Match Ratio	Test Status
SL090200	Lodging accommodations	Commodity level	90111502	Lodges or resorts	0.5	Failed
SW117218	Warehouse storage - storage of refrigerated goods	Commodity level	78131801	Refrigerated storage	1	Passed
SW056000	Cloud Services – platform as aservice(PAAS)	Class Level	81162100	Cloud based platform	0.66	Passed

Improvement scope

The accuracy of the project can be increased by trying to process all the text in the same form of speech, so words like “lodging” can be matched with “Lodge”. This will require a few additional NLTK libraries to be added to the project. Come up with a logic to match the synonyms words, As the UNSCPC code can be at any 4 levels, and if there exists an Avalara category that does not have meaningful match at the commodity level, but there might be a general category at Segment level which can be categorized same as Avalara category. As the longest common subsequence is applied to words, the complexity is already at its optimized state with accuracy and time. There is a huge scope of improvement in the text preprocessing aspect of the project.

Reference

1. Inflect engine: https://pypi.org/project/inflect/
2. NLTK library: https://www.nltk.org/
3. Anytree: https://anytree.readthedocs.io/en/latest/
4. Observable: https://observablehq.com/