IIT Madras
Faculty of Engineering

[CH5019] Mathematical Foundations of Data Science

Teacher:

Group Project — Jan-May 2022

Due date: June 3rd, before midnight

Contents

1 Group 16 - Team Members

NAME	ROLL NO.
Sriram S M	ME19B051
Vaibhav Mahapatra	ME19B197
K J Sashank	EE19B137
Kalash Verma	ME18B052

2 Used Software and Tools

Programming Language: Python
Modules:

- ➤ Natural Language Toolkit - to process text and break down paragraphs to sentences and words
- ➤ RE - to use RegEx (redular expressions) to filter strings
- ➤ Numpy - to perform vector operations and quanitfy similarlity using various metrics like cosine similarity, Manhattan distance

3 Learning Outcomes

Working on this group project helped our team develop a wide variety of skills. These skills range from technical skills in the Data Science and Natural Language Processing space to soft management skills attained from coordinating and working in a team. They can be summarised as follows:

Technical Skills

- ➤ Using *nltk* for text processing, from tokenization and lemmatization to removing stopwords. Understood the impact of each text processing step on the accuracy of the final result.
- ➤ Using *re* to filter and process strings using Regular Expressions. Learnt how RegEx can be compactly used to identify and segment sections of text as per need.
- ➤ Using *numpy* to vectorize the process text into numeric forms and mathematically determine the similarity between two vectors. Learnt the implications of using various metrics like cosine similarity, Manhattan distance, etc. and the trade-offs involved in using them for our use case.

Soft Management Skills

- ➤ Teamwork: Learnt the importance of communicating and coordinating proactively.
- ➤ Communication: Understood the impact of communicating effectively and regularly on mapping the progress of the project and planning accordingly.
- ➤ Planning: Acknowledged the importance of laying down a structured plan with specific, measurable and time-bound goals.

4 Understanding the Problem Statement

An instructor wants to grade answers to descriptive questions automatically. The instructor has a template best answer that she/he has developed and wants to use the same to grade descriptive answers of students. The objective here is to write code which compares a student's answer with the ideal answer template and grade the answer. The comparison results between the template and the student provided answer could be categorical (right vs wrong) or continuous (75% similarity and so on). This can be approached broadly in a sequential manner as follows:

1. Make features by understanding what characterises an answer
2. Learn how to represent text in numeric or vector formats to numerically compare 2 strings
3. Understand various comparison metrics and their trade-offs to effectively benchmark 2 string vectors

After developing a holistic approach to the above steps, we can apply them to multiple test cases and see if they give a satisfactory result for our use case.

5 Our Solution

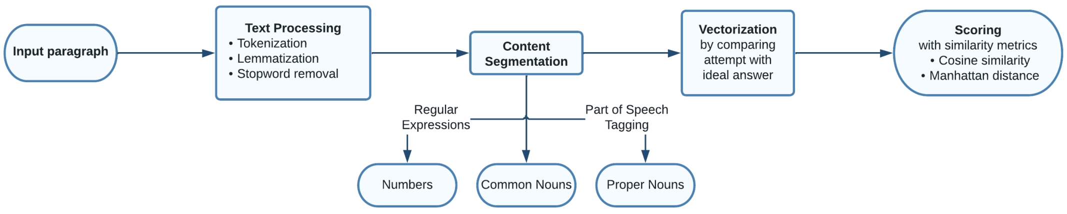
5.1 Introduction

Descriptive questions are of several types. Some questions are fact and concept based which test recall and reasoning skills of a student. Questions that asks to describe, define or explain something as well as fact based questions that asks what, when, who type of questions comes under this category. These questions generally have just one correct answer semantically or at most have finite number of different possible answers.Other type of questions are more open-ended and can have many different correct answers. These type of questions can asks for opinions of a student and therefore answer will differ for each student.In this assignment we are focused on first type of question described above. Evaluating them is easier than as they are fact based and we can find directly compare with the ideal answer to evaluate the attempted answer. For comparing the answers we have split the words in the answer into following subsets:

- Set of Proper Nouns mentioned in the answer
- Set of Numbers mentioned in the answer
- Other Common nouns and words that are not stop-words

5.2 Workflow Process

The workflow is visible in the below flowchart. Our approach for each step is also explained in subsections subsequently:



5.2.1 Text Processing

Tokenization Tokenization as name implies involves breaking down of raw text into small chunks or tokens. Tokens can be sentences or words. These tokens are then used to build NLP models. When text is seperated into sentences it is called sentence tokenization and when text is split into words it is called word tokenization. We have tokenized the answer text into sentences using python's nltk module. Then used Regex for word tokenization. The below code segments summarise how we implemented this step.

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
import re

# Sentence Tokenization
sentences = nltk.sent_tokenize(ideal)
for sentence in sentences:
    print(sentence)
    print()

# Word Tokenization with RegEx
word_list = []
for sentence in sentences:
    words = re.sub("[^a-zA-Z]"," ",sentence).split()
    print(words)
    print()
    word_list.extend(words)
```

Lemmatization

Lemmatization is a text normalization method that is used to prepare text, words and documents for further processing in the Natural language processing domain. It considers a language's full vocabulary to apply a morphological analysis to the words,for example - the word "running" is reduced to it's lemma "run". This provides lesser number of root lemmas to be stored in the dataset making the further work faster. Lemmatization also takes care of grammatical corrections to root words such as when "ran" is the input word to be lemmatized, it is reduced to it's lemma "run" as well. This helps while comparing test answers with the template answer as the test answer may contain a lot of grammatical variations of the given word in the template and hence the final result is accurate. We used the classical WordNet Lemmatizer to normalize the text.

```
import nltk
from nltk.stem import WordNetLemmatizer

# Using WordNet Lemmatizer for Lemmatization
lemmatizer = WordNetLemmatizer()
lemm_words = [lemmatizer.lemmatize(word, pos=wordnet.VERB) for word
    ↪ in word_list]
```

Stopword Removal

Stopwords are words that do not add much information to the text. These words are in abundance in language and can be removed (depending on application) to focus on more important information. Some examples of Stopwords are is, are, would, The, who, what etc. These words usually skew the frequency analysis as stopwords appear in large frequencies. Hence, we used wordnet's set of stopwords to filter them out.

```
    import nltk
    from nltk.corpus import wordnet

    # Removing stopwords from a list of words
    word_list = [word for word in word_list if word not in set(stopwords.words('
        ↪ english'))]
```

5.2.2 Content Segmentation

To calculate similarity score we have segmented the set of words obtained after text processing into proper nouns, numbers and common nouns. This is done because we believe that proper nouns and number add more information than common nouns.

Proper Nouns using Part of Speech Tagging
We have used the part of speech tagging method of nltk package to find proper nouns.

```
import nltk

ppn_list = []
tagged = nltk.pos_tag(lemm_words)
for (word, tag) in tagged:
    if tag == 'NNP': # If the word is a proper noun
        ppn_list.append(word)
```

Numbers Using RegEx
We have used Regex to seperate numbers.

```
import re

numbers = re.sub("\D","␣",ideal).split()
```

Common nouns and others
The remaining words are collected. For these words we also perform synonym matching for attempted answer using wordnet database.

```
import ntlk
from nlk.corpus import wordnet

cmn_list = [word.lower() for word in list(set(lemm_words) - set(ppn_list))]

# Synonym Matching using wordnet synsets
for word in ideal_cmn_list:
        for syn in wordnet.synsets(word):
            for i in syn.lemmas():
                for k in range(len(cmn_list)):
                    if i.name() == cmn_list[k]:
                        cmn_list[k] = word
```

5.2.3 Vectorization

In order to perform operations on any data, the model generally requires it to be in numerical form. The idea behind vectorization is to converted the alphanumerical data that was obtained as a result of the preceding operations and convert it to some form of numerical data appropriate to our purpose.

In our case we use the method popularly known as 'term frequency' in natural language processing. A corpus of words is formed from the sets of words in the processed template and student's attempt. A list of the no.of occurrences of each word in the corpus is made for each of the processed template answer and the student's answer. So we end up with 2 lists, with no.of elements equal to the size of the corpus and each entry being a whole number. The implementation for the same can be seen below:

```
def vectoriser(ideal, attempt):
    word_set = list(set(ideal).union(set(attempt)))

    word_count_ideal = {}
    word_count_attempt = {}
    for word in word_set:
        word_count_ideal[word] = 0
        word_count_attempt[word] = 0
    for word in ideal:
        word_count_ideal[word] += 1
    for word in attempt:
        word_count_attempt[word] += 1

    return list(word_count_ideal.values()),list(word_count_attempt.values())
```

5.2.4 Scoring

Finally, the 2 numerical vectors are compared using similarity computing techniques and the student's attempt is evaluated.

Cosine Similarity

Cosine of the angle between the 2 vectors is calculated by dividing their dot product by their magnitudes. The closer the vectors are, higher is the cosine of the angle and hence higher is the similarity between the 2 answers.

```
import numpy as np
from numpy.linalg import norm

def scoreit(ideal_vec, attempt_vec):
    sim = np.dot(ideal_vec, attempt_vec)/(norm_i*norm_a)
    if sim != 0:
        sim = sim / (norm(ideal_vec), norm(attempt_vec))
    return sim
```

Manhattan distance

Manhattan Distance is also called L1 distance between two vectors. Mathematically, we can compute the Manhattan distance between two n-dimensional vectors **x** and **y**:

$$\sum_{i=1}^{n} |x_i - y_i|$$

This metric can work in our frequency analysis as each element of the word vectors is either 0 or 1. The best answer will have 0 Manhattan distance whereas the worst answer will have Manhattan distance = length of the ideal vector. Manhattan distance has it's disadvantages when used as a metric for a learning based algorithm. Hence, we decided to focus on cosine similarity.

Weightage

Depending on the context, answers may need to be evaluated with unequal weights on proper nouns, common nouns, numbers, etc. These weights, which add up to 1, are assigned manually based on the context, multiplied with the corresponding similarity score and are then added to obtain the final score. Weightage can be treated as a Hyperparameter that depends on our use case.

6 Conclusion

In the assignment we have implemented a simple model to grade descriptive answers of a student as explained above. Simplicity of the model makes it computationally inexpensive and fast. It can also handle synonyms. The above model works very well for fact based, define and explain type question. However,the model is very naive and doesn't take into account several cases. It won't perform well for questions that can have multiple different answers as explained in introduction. Removal of stop words while generally removing low level information can also cause issues. For example just addition of word 'not' can completely change the meaning of the sentence. The issue with the stop word removal can be handled relatively easily using rule based methods. However handling opinionated questions is difficult. While an Ideal answer can be given as a sample for these type of questions these answers will rarely if ever contain all information necessary to analyse attempted answer. Semantic analysis methods can be used to improve accuracy for first type of questions(fact based) and also handle the other type of questions.

7 Next Steps to take this project forward

As mentioned in the conclusion the above model is simple and works for very specific type of scenarios. There several steps that can be taken to improve upon our model such as using DL vectorisers, using embeddings such as word2vec etc. We can also use a dataset of past student answers for questions, ideal answers and score they got to train a ML model that can map similarity scores to marks obtained. Some of the other metrics that can be used to solve polysemy and context mapping will be to use Latent Semantic Analysis on the given vectors which is basically doing principal component analysis in the NLP domain. Also, multigram model approach and ESA wordnets can be used to solve context matching - overall these metrics will help improve and develop the project.

8 References

1. Intro to NLP: https://towardsdatascience.com/introduction-to-natural-language-processing-for-text-df845750fb63
2. Similarity metrics: https://medium.com/dataseries/similarity-and-distance-metrics-for-data-science-and-machine-learning-e5121b3956f8
3. Synonym Matching: https://www.holisticseo.digital/python-seo/nltk/wordnet
4. https://www.askpython.com/python/examples/tf-idf-model-from-scratch
5. https://medium.com/@adriensieg/text-similarities-da019229c894
6. https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4
7. https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a

9 Contribution Report

NAME	ROLL NO.	CONTRIBUTION.
K J Sashank	EE19B137	Ideation, Text processing theory and code structure implementation, making test cases
Vaibhav Mahapatra	ME19B197	Ideation, improving text processing, structuring content segmentation, researching similarity metrics, writing report
Sriram S M	ME19B051	Ideation, Implementing Synonym matching, vectorising and implementing similarity metrics, beautifying code
Kalash Verma	ME18B052	Ideation, literature review and analysis of model, report drafting, making test cases