# NLP Assignment 1: Text pre-processing and matching using spaCy

--Utsav Basu (20CS30057)

## Problem Statement

The task is to make a retrieval system for Quora. Given a new question (query), we want to find out the most similar questions that already exist (documents) in the Quora database, so that they can be shown as recommendations. This assignment is meant to use Python and spaCy which is a library for performing several NLP related tasks easily.

## Dataset

The dataset 'Query_Doc' contains 3 CSV files.

- The 'query' file contains 100 query ids and query texts.
- The 'docs' file contains 10,000 doc ids and doc texts.
- The 'qdrel' file contains the query ids and the doc ids, if they are similar. 130 such relations are present in the file. Use this for evaluating the methods.

## Methods and Results

### Pre-procesing

For preprocessing the docs and queries, the "contextualSpellCheck" library is used. It is a spaCy extension, which performs spell-checking. In addition, the characters other than alphanumeric and whitespaces are removed. This pre-processed text is further processed in further parts to perform the required tasks.

### Task 1: Tokenization

The docs and queries are tokenized using spaCy library. Their TF-IDF vectors are calculated using sklearn library, and then the Precision@k scores for k = 1, 5, 10, was determined. The results obtained are as below:

|  |  |
|---|---|
| **Vocab size** | 2273 |
| **Average Precision@1** | 0.5400 |
| **Average Precision@5** | 0.7700 |
| **Average Precision@10** | 0.8155 |

## Task 2.1: Stemming

Stemming is performed on the docs and queries using nltk library. Their TF-IDF vectors are calculated using sklearn library, and then the Precision@k scores for k = 1, 5, 10, was determined. The results obtained are as below:

|  |  |
|---|---|
| **Vocab size** | 2011 |
| **Average Precision@1** | 0.6200 |
| **Average Precision@5** | 0.8005 |
| **Average Precision@10** | 0.8415 |

## Task 2.2: Lemmatizing

Lemmatizing is performed on the docs and queries using spaCy library. Their TF-IDF vectors are calculated using sklearn library, and then the Precision@k scores for k = 1, 5, 10, was determined. The results obtained are as below:

|  |  |
|---|---|
| **Vocab size** | 1997 |
| **Average Precision@1** | 0.6400 |
| **Average Precision@5** | 0.7955 |
| **Average Precision@10** | 0.8328 |

## Analysis

As we observe for both stemming and lemmatization the scores increase. This is obvious, since instead of considering the word as a whole [‘add’, ‘addition’, ‘adder’, etc.] we only consider the root forms [‘add’]. Although for NLP related tasks, lemmatization should perform better, however in our retrieval system, as we observe, stemming performs better. This is expected, since while writing queries, people tend to just throw a bunch of words best describing their intent. Thus, a stemming performs better here. Although, to be noted, both performed better than the tokenizer baseline.

## Task 3: NER and POS tagging

NER and POS tagging is performed on the docs and queries using spaCy library. Their TF-IDF vectors are calculated using sklearn library, and then the Precision@k scores for k = 1, 5, 10, was determined. The results obtained are as below:

| | |
|---|---|
| **Vocab size** | 2190 |
| **Average Precision@1** | 0.5500 |
| **Average Precision@5** | 0.7980 |
| **Average Precision@10** | 0.8215 |

### Analysis

We observe, though the scores are better than the baseline, they are not as good as the stemmer or lemmatization scores. One possible explanation is that, although names entities and POS tags were applied to the docs and queries, it is possible that similar entities end up as completely different entities, while different entities are getting recognized as same. For example, “Indians” and “Indiana Jones” are different, but maybe recognized as similar entities (for the “Indian” part), while, “Virat” and “Kohli” although similar, might end up as different entities. Thus, a lack of context is the primary reason of such inaccuracies. Maybe a machine learning solution for

## Task 4: My Method to improve performance

One observation for the TF-IDF vectors was that for each vector, most of the entries were 0. This made sense, since, all the words in the vocabulary wouldn’t, of course, occur in all the docs and queries. As a result, these words don’t contribute anything to the cosine-similarity scores. To avoid that, a Gaussian Noise ($\mu = 0, \sigma = 0.01$) of added to all the vectors. Since some randomness is involved the mean over 100 runs is reported here:

| | |
|---|---|
| **Mean Vocab size** | 1951.0 |
| **Mean Average Precision@1** | 0.6109 |
| **Mean Average Precision@5** | 0.8183 |
| **Mean Average Precision@10** | 0.8492 |

### Analysis

It can be observed that the scores are much better than all of the above methods. This is because having a non-zero contribution from all the words is much better than simply none. Thus, adding some random noise, accounts for the words that the user skipped, resulting in a better score.