**Documentation**

The solution has been implemented in python Jupyter notebook in a separate virtual environment. You can skip the Step 1 and 2 if you do not intend to play with the code.

Step 1: Create a virtual environment using virtualenv

$ python -m venv python\_env

To activate the virtual environment

$ python\_env\Scripts\activate

Step 2: Install the dependency packages specified in requirements.txt

$ pip install -r requirements.txt

**To open the notebook, execute below command.**

**$ jupyter notebook**

Incase if running the notebook is a hiccup, I have also attached .PDF format of the notebook

**Solution:**

**About Data:** Dataset consists of English sentences consisting of alphabets, numbers, punctuations, and few special characters.

**Cleaning Text:** To clean the text I am performing below preprocessing step,

* Add Extra space before and after .,!?
* Except for alphabets (lower & capital), numbers, .,!'? replace other characters with spaces
* Remove extra and trailing spaces

Once the text is clean, I will derive pos-tags for each word in the sentences. This way we can extract more information from the text.

Now if we combine each word with its pos-tag, we will be able to differentiate between the same word occurring in different contexts.

**Transformation:** I have used Tf-Idf vectorization method to transform the text into numeric vectors. But the Tf-Idf representation is sparse and requires more computation power.

While transforming text into tf-idf vectors I considered unigrams in each sentence. Also, if a word occurs in more than 95% of sentences, I am ignoring it.

So, I applied PCA – dimensionality reduction technique to convert them into dense representations. I preserved 95% of variance which left me with 1256-dimensional vector.

**Similarity Calculation:** I am using cosine similarity as a metric to measure the similarity between two sentences. Its value will be ranging from [0, 1] where 0 being two sentences are dissimilar and 1 being highly similar.

As per the requirement we need to find out top 3 similar sentences in the dataset for each sentence. I need to compare each sentence with every other input sentence, but we don’t need to store the similarity scores of all sentences. At any point of time we only need the top 3 similar sentences.

Hence, I have used a heap to keep track of the top – 3 similar sentences and their scores for every sentence. This saves a lot of space.

Also I optimized the code by avoiding repetitive calculations. Since cosine similarity is a symmetric metric, CosineSimilarity(A,B ) = CosineSimilarity(B, A) I skipped the second calculation saving few iterations.

I have set the similarity threshold value to 0.5.

**Further improvements:**

I worked on a low configuration machine, but with the availability of better configuration we can extend the Tf-Idf vocabulary to bigrams.

Beyond these, we can also create embedded representations using Word2Vec or Glove or Language Models for each sentence instead of using Tf-Idf and NMF based vectors. But if we consider embeddings, instead of using cosine similarity we need to use “angular cosine similarity” as metric.

def angular\_cosine\_similarity(A, B ):

return 1 – cosine\_similarity(A, B ) / PIE