HGP Internship Assignment

Name : Abhishek kumar Singh
Email : iabhikmr2000@gmail.com
Github : AbhiSaphire
Portfolio : Abhishek Portfolio

According to given Problem Statement:
Given an input string from a user, I need to parse it into components to be used for further processing.
These components will be best matches against predefined lists and / or scalars.

My Approach to problem
Let's divide and conquer all todos separately.

1. For **Separating/Selecting the Sectors** from the input string. I would suggest using LDA (Latent Dirichlet Allocation) for Topic Modelling. Because it is fast and very powerful especially in these types of scenarios.

   Preprocessing Stage would require -
   - **Tokenization** - This will split text into sentences, and sentences into words. Then lowercasing the words and removing punctuations.

     ```
     from nltk.tokenize import PunktSentenceTokenizer, word_tokenize
     def word_sentence_tokenize(text):
      sentence_tokenizer = PunktSentenceTokenizer(text)
      sentence_tokenized = sentence_tokenizer.tokenize(text)
      word_tokenized = list()
      word_tokenized.append(word_tokenize(token) for token in sentence_tokenized)
      return word_tokenized
     ```

   - **Stopwords Removal**
     ```
     review = re.sub('[^a-zA-Z0-9]', ' ', text)
     review = review.lower()
     review = review.split()
     ps = PorterStemmer()
     review = [ps.stem(x) for x in review if not x in set(stopwords.words('english'))]
     review = ' '.join(review)
     ```

- **Lemmatization** - This will change any word from 3rd person to 1st person or any verb from past or future tenses to present.
- **Stemming** - This step will change words to their root forms.
  For all preprocessing mentioned above we can simply use NLTK and gensim libraries.

```
import gensim
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
# nltk.download('wordnet')
def lemmatize_stemming(text):
        stemmer = PorterStemmer()
        return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

def preprocess(text):
        result=[]
        for token in gensim.utils.simple_preprocess(text) :
                if token not in gensim.parsing.preprocessing.STOPWORDS:
                        result.append(lemmatize_stemming(token))
        return result

print(preprocess("Output Revenue, EBITDA margin for Steel and Metal stocks for past 10 qtrs"))
```

Outputs Preprocessed Data

```
['output', 'revenu', 'ebitda', 'margin', 'steel', 'metal', 'stock', 'past', 'qtr']
```

Bag of Words conversion -

```
dictionary = gensim.corpora.Dictionary(sectors_docs)
bow_corpus = [dictionary.doc2bow(doc) for doc in sectors_docs]
```

Applying LDA -

Hopefully LDA would be able to separate Sectors and Fundamentals.

```
lda_model =  gensim.models.LdaMulticore(bow_corpus, num_topics = 2, id2word = dictionary, passes = 10, workers = 2)
```

2. For separating **Time Period and Unit of time period** we can use regex. This step should be used before preprocessing because preprocessing will most probably stop the digits.

```
import re
string = "Output Revenue, EBITDA margin for Steel and Metal stocks for past 10 qtrs"
match = re.search("\d+\s*\w+", string)
print(match)
```
Outputs: `<re.Match object; span=(66, 73), match='10 qtrs'>`

3. For handling **Contextual Similarity** between words we can use WordNet's synsets which gives synonyms of the given word and check it with words from Sectors_doc for cosine distance between words and return the most appropriate word whose score is highest.
   This is a Pseudo code for checking contextual similarity between two words and returning True if words are more than 70% similar.

   ```
   from nltk.corpus import wordnet as wn
   from itertools import product

   def contextual_similarity():
       wordx, wordy = "revenue","sales"
       sem1, sem2 = wn.synsets(wordx), wn.synsets(wordy)
       maxscore = 0
       for i,j in list(product(sem1,sem2)):
           score = i.wup_similarity(j)
           maxscore = score if maxscore < score else maxscore
       return True if maxscore > 0.70 else False

   print(contextual_similarity())
   ```

   Outputs:
   ```
   True
   ```

4. For Handling **Syntactic Similarity** -
   - **"qtrs", "quarters" should match / "years", "yr"**. These types of problems can be handled by lemmatization and stemming, which was a part of preprocessing we did. Every word will be changed to its root form.
   - Handling spelling mistakes would be a big task here, the best I know is to use SymSpell library. The only trick here is SymSpell is a c# library but using it in python is not that difficult.

```
from collections import Counter
from sklearn.datasets import fetch_20newsgroups
import re
corpus = []

for line in fetch_20newsgroups().data:                           #This is a Pseudo Code
        line = line.replace('\n', ' ').replace('\t', ' ').lower()  #Here newspaper corpus is made
        line = re.sub('[^a-z ]', ' ', line)                        #Just to show how SymSpell works
        tokens = line.split(' ')                                   #While building real model we can
        tokens = [token for token in tokens if len(token) > 0]     #use suitable corpus
        corpus.extend(tokens)
corpus = Counter(corpus)
corpus_dir = '../'
corpus_file_name = 'dorian_gray.txt'
symspell = SymSpell(verbose=10)
symspell.build_vocab(dictionary=corpus, file_dir=corpus_dir, file_name=corpus_file_name)
symspell.load_vocab(corpus_file_path=corpus_dir+corpus_file_name)
results = symspell.correction(word='helol')
print(results)
```

Entire Process -

1. Use regex to separate Time period and Unit of time period and split them to store them separately.
2. Preprocess the Data.
3. Separate Sectors by topic modelling using LDA, again use topic modelling to get sub-topics or fundamentals with corpus of fundamental_docs.
4. Use Contextual similarity and Syntactic similarity algorithms of separated Sector names and fundamental name (for proper spell check and synonym check).
5. Store Sector, Fundamental, Attributes of Fundamentals, Time Period, Unit of Time period as keys in a dictionary and append their values. Return dictionary!

I hope I was able to deliver proper solutions to the problems. I am sure there are many more techniques to tackle mentioned problems but due to time constraint, I can only think of provided solutions. Sorry for the delay (due to my college’s surprise online internal exams). Looking forward to your response.