Requirements for working with non-english corpuses

Non-English documents and their parsing and reduction for use in the Singular Value Decomposition (SVD) Variational Autoencoder requires several additions and extensions to the codebase.

First, tokenization of paragraphs into sentences will need to be added as at present this task is achieved simply by hand coded regular expression punctuation and symbol filtering permitting sentences to be separated by periods. (see the sections ([1] and [2]) of 'corpus2sentences.py' and section [1] of 'filter_sentences.py', both below, for example)

Non-English sentences will need further language specific filtering.
Therefore the use of the 'sent_tokenizer' from the (already imported) Python 'nltk.tokenizer' module will be required. This sentence tokenizer takes a second argument specifying the target language of the document being split. This would best be placed in added section [2b] of 'corpus2sentences.py'.

Second, following the tokenization of paragraphs into sentences, also at [2b], a language specific tokenization of sentences into words would be required.

Third, for each language in the spectrum of corpuses a corresponding set of 'stopwords' (common words with no semantic distinctions such as English 'the', 'an', 'as', etc.) would need to be supplied and filtered out. This is fairly simple because the Python 'nltk' (natural language toolkit) already has stopwords for many languages. All that would be needed is to specify the language in 'stop_words = stopwords.words(<language>)' where at present the argument is 'english'.

Fourth, similarly stemming (the removal of tense and person-case prefixes and suffixes) would also have to be added for each desired language. At present 'PorterStemmer' is used in 'filter_sentences.py' at [4] to stem English words. Another possible stemmer is 'nltk.stem.SnowballStemmer' which can stem 13 languages, and there are others as well.

Fifth, the final stage before forming the sentence-term matrix is to convert each word in every sentence into a frequencey of usage in the entire corpus. This is done as seen below by the 'scikitlearn' object 'TfidVectorizer' which requires a tokenizer for each specific language. The code for 'sentences2matrix.py' below shows the use of the default 'TfidVectorizer' which uses an English tokenizer. An alternative would look like the following:

```
vectorizer = TfidfVectorizer(ngram_range=(1,1), tokenizer=jap_tokenizer)
```

Following these changes and additions non-English languages seem to be able to be accomodated given the availability of software to perform each of the four steps mentioned for each language corpus desired.

Further, most non-English languages will require strings to be unicode due to the presence of specific non-ascii characters. All modules used in the Variational Encoder will need to accommodate unicode string processing and storage.

An example of possible resources for a relatively difficult case, Arabic, is provided below. The most difficult aspect is that if the word tokenizer provided in ArabicProcessingCog failed or was ineffective a TfidfVectorizer substitute would need to be built by hand using the Arabic 'wordfreq' module available via the pypi project referenced below.

github.com/disooqi/ArabicProcessing Cog (tokenization into sentences and into words)
github.com/mohataher/arabic-stop-words/blob/master/list.txt (stop words)
github.com/mohazsaad/arabic-light-stemming.py (stemming)
pypi.org/project/wordfreq (used to build hand-built TfidfVectorizer substitute)

specific change references in the Variational Autoencoder modules

 # corpus2sentences.py

```
for root, dirs, filenames in os.walk(basepath):
    for fn in filenames:
        if fn.endswith(".txt"):
            with open(basepath+fn, "r") as f:
                s = ''  #string representation of file f
                for line in f:
                    if not line.startswith('$'):
                        s += line.replace("\n"," ").lstrip()


                #filter each doc:
                print('\nfiltering text ' + fn )

                #[1] eliminate i.e, e.g., A. Taylor
                rs = '\s([a-z,A-Z]\.)+,*'
                regex = re.compile(rs)
                s_pf = s
                s = filter(regex, '', s_pf, diagnostics)

                #[2] eliminate citations of form '[34]' for example
                rs = '\[\d+\]'
                regex = re.compile(rs)
                s_pf = s
                s = filter(regex, '', s_pf, diagnostics)


                #split the file-string on '***' to form docs
                a = s.split('***')
                if diagnostics: print(str(len(a)) + ' documents')

                for doc in a:
                    doc_sentences = []  #sentences associated with docs[doc]
                    sentences = doc.split('.')  #sections of split doc string
```

 # filter_sentences.py

```
#[1] filter out punctuation
#NOTE: string.punctuation for punctuation in sentence.translate WORKS!
punctuation = str.maketrans({key: None for key in string.punctuation})
for k,paragraph in docs.items():
    paragraph = [sentence.translate(punctuation) for sentence in paragraph]
    _docs[k] = paragraph

#[2]tokenize each sentence to a temporary word list for [3],[4]
#load stopwords
# _docs is dictionary of paragraphs - values are arrays of tokens
stop_words = stopwords.words('english')
for k,paragraph in _docs.items():
    a = []

    [2b] may need language specific sentence and word tokenization
    for sentence in paragraph:
        sentence += sentence.lower()        #lower case
        wordlist = word_tokenize(sentence)  #[2]tokenize ea se`nt in ea pgph

        #[3]filter out stopwords
        wordlist = [word for word in wordlist if word not in stop_words]
        if diagnostics == True:
            print('\n' + str(wordlist))

        #[4]stem each word in each word list
        porter = PorterStemmer()
        wordlist = [porter.stem(word) for word in wordlist]
        a.append(wordlist)
        if diagnostics == True:
            print('\n' + str(wordlist))

    #store in _docs
    _docs[k] = a
```

 # sentences2matrix.py

```
#create dataframe
df1 = pd.DataFrame(sentences)


# Initialize TfidfVectorizer - create sentence-token matrix
vectorizer = TfidfVectorizer()
doc_vec = vectorizer.fit_transform(df1.iloc[0])
df2 = pd.DataFrame(doc_vec.toarray().transpose(),
      index=vectorizer.get_feature_names())
df2.columns = df1.columns
```

In summary, accomodating non-English languages would require major additions and changes to the Variational Autoencoder, some of them depending on untested language variations in 'nltk' natural language processing (NLP) modules, and some requiring custom processing modules and language data. Support for major European languages would most likely be feasible, but support for non-European languages would most likely be very difficult and require custom modules to be built per language. In addition, almost all languages would require unicode strings so that all Python string and NLP processing modules would need to be tested and/or verified to adequately support unicode operations throughout.