Wikipedia-TF-IDF

Steps

1. Choose any random wikipedia article as a seed page (Supervised Learning (https://en.wikipedia.org/wiki/Supervised_learning))

2. Gather all the immediate outlinks from the page and form a set of these outlinks

```
from goose import Goose

url = 'https://en.wikipedia.org/wiki/Supervised_learning'
g = Goose()

article = g.extract(url = url)

outlinks = article.links
```

3. Curate the links, form a set and store in a pickle file.

4. Load the pickle file (there are roughly 127 links in the set).

```
import pickle

links = pickle.load(open('wiki-links.p', "rb"))
```

5. Crawl through each of the page and store the cleaned text from each page in a separate text file.

```
from goose import Goose
g = Goose()

for link in links:
 url = "https://en.wikipedia.org" + link
 article = g.extract(url=url)

 # Files are stored in a separate folder named "Wikipedia-Pages"
 filename = "Wikipedia-Pages/" + link[6:]

 f = open(filename, 'w')
 f.write(article.cleaned_text.encode('ascii', 'ignore'))
 f.close()
```

6. Now we create an RDD that loads all these files and stores them in a (key, value) pair format where key is the path of the file and value is the contents of the file. It is

important that contents of files should not get mixed up, hence we use `wholeTextFiles` from SparkContext. Further we need only the content and not the paths, hence we extract only values and finally we split the text of each article to obtain only words.

```
documents = sc.wholeTextFiles("Wikipedia-Pages/").values().map(lambda doc: r
e.split('\W+', doc))
```

7. Now we move onto creating the Term Frequency (TF) for every word in each document. This can be done by using the `HashingTF` class from `mllib.feature` package of Spark.

```
from pyspark.mllib.feature import HashingTF

hashingTF = HashingTF()
tf = hashingTF.transform(documents)
```

8. In the next and the final step we compute the IDF for every word in all the files and scale the TF obtained in the previous step by the IDF.

```
from pyspark.mllib.feature import IDF

tf.cache()
idf = IDF().fit(tf)
tfidf = idf.transform(tf)
```

9. Save the awesome TF-IDF result of every page in an output folder. Multiple files will be created based on the partitioning of the RDD.

```
tfidf.saveAsTextFile("tf-idf-output")
```

10. The final output is stored in the `tf-idf-output` folder. The output file shows tf-idf vector for each page as a separate tuple. The second entry in the tuple is the list of words (encoded using a hashing function) that occur in that particular page and the third entry in the tuple is the list of tf-idf values for those words respectively.