Ex:No:13

NLP TASK: PART OF SPEECH TAGGING & INFORMATION RETRIEVAL.

AIM:

To perform part of speech (pos) tagging on a given text using spacy and develop an information retrieval system that sorts a set of documents based on their relevance to be a user query using tf-idf vectorisation and cosine similarities.

Algorithm:

1> Load spacy English model and input that text of pos tagging.

2> process the text to tokenize and assignment of speech tags to each token.

3> collect documents and a query. then combine them into a single corpus.

4> vectorize the corpus using TF-Idf and compute cosine similarity between the query.

5> Rank the documents based on similarity scores and display the most relevant ones.

OUTPUT:

```
AI        -> PROPN
-         -> PUNCT
driven    -> VERB
platforms -> NOUN
Learning  -> VERB
Paths     -> NOUN
And       -> CCONJ
help      -> VERB
students  -> NOUN
grasp     -> VERB.
concepts  -> NOUN
faster    -> ADV
.         -> PUNCT.
```

Program.

```
import spacy
from sklearn.feature-extraction text. import Tf
IdfVectoriser.
from sklearn. metrics. pairwise import cosine.
                                          similarity.
NLP = Spacy.load("en- core- web- sm")
text = "AI - driven platforms personalise learning
paths and helps students grasp concepts faster".
    doc = NLP (text)
    print ("part - of - Speech.Tagging:\n")
    for token in doc:
        print(f"token- text:15} -> {token:
                                     pos-})
documents = [
    "AI tools analyse student performance and
provide real- time feedback".
    "Intelligence autoring systems adapt to
each student learning style'.
    "A'I helps automate grading and administ
                           tasks in schools. -tive
    "chatbots assists student with
answering questions any time of days".
```

```
]
query = "How does A I support student in
                              learning";
  corpus = documents + [query]
  vectoriser = If idf vectoriser
  If id - matrix = vectoriser . fit = transform [corpus]
  similarities = cosine - similarity [ tf.idf - matrix
                              [-1] , tf id - matrix
                             [-1] flatten ().
  ranked = docs = sorted (zip (similarities , documents)
reverse = True)
  print ("\n top relevant documents:\n")
for score , doc in ranked - docs :
   print (f" score = {score :2f} -> {doc}")
```

Result:

Thus the python program for NLP has been executed successfully.

[illegible]