2403A52031

AkULA MANOJ

LAB-ASSIGNMENT-8.4

```
import re
import math
import numpy as np
import pandas as pd
from collections import Counter
```

DATASET CREATION

```
corpus = """
Natural language processing is an important area of artificial
intelligence.
It allows machines to understand and generate human language.
Language models predict the likelihood of word sequences in sentences.

Artificial intelligence is transforming education and technology.
Students use intelligent tools for learning and productivity.
AI systems help automate tasks and improve efficiency.

Technology impacts society by improving communication and automation.
Ethical considerations such as bias and fairness are important.
Responsible AI ensures trust and safety.

Language models are used in chatbots search engines and translation
systems.
Education systems are evolving with artificial intelligence tools.
Students benefit from personalized learning experiences.
""" * 35   # ensures more than 1500 words

print("Total words:", len(corpus.split()))
```

```
Total words: 3500
```

TEXT PREPROCESSING

```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = text.split()
    return tokens
```

TRAIN / TEST SPLIT

```
tokens = preprocess_text(corpus)

split_index = int(0.8 * len(tokens))
train_tokens = tokens[:split_index]
test_tokens = tokens[split_index:]

print("Training tokens:", len(train_tokens))
print("Testing tokens:", len(test_tokens))
```

```
Training tokens: 2800
Testing tokens: 700
```

BUILD N-GRAM MODELS

```
def generate_ngrams(tokens, n):
    return [tuple(tokens[i:i+n]) for i in range(len(tokens)-n+1)]

unigrams = generate_ngrams(train_tokens, 1)
bigrams  = generate_ngrams(train_tokens, 2)
trigrams = generate_ngrams(train_tokens, 3)

uni_count = Counter(unigrams)
bi_count  = Counter(bigrams)
tri_count = Counter(trigrams)

vocab_size = len(set(train_tokens))
```

FREQUENCY TABLES

```
pd.DataFrame(uni_count.most_common(10), columns=["Unigram", "Count"])
```

```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Unigram\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          [\n            \"of\"\n          ],\n
[\n            \"language\"\n          ],\n          [\n
\"are\"\n          ]\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Count\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 50,\n        \"min\": 56,\n        \"max\": 224,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          112,\n
56,\n          224\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
pd.DataFrame(bi_count.most_common(10), columns=["Bigram", "Count"])
```

```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Bigram\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          [\n            \"area\",\n
```

```
\"of\"\n          ],\n          [\n            \"language\",\n
\"models\"\n          ],\n          [\n            \"is\",\n
\"an\"\n          ]\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Count\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 18,\n        \"min\": 28,\n        \"max\": 84,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          84,\n
56,\n          28\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
pd.DataFrame(tri_count.most_common(10), columns=["Trigram", "Count"])
```

```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Trigram\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          [\n            \"artificial\",\n
\"intelligence\",\n            \"it\"\n          ],\n          [\n
\"language\",\n            \"processing\",\n            \"is\"\n
],\n          [\n            \"important\",\n            \"area\",\n
\"of\"\n          ]\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Count\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 28,\n        \"max\": 28,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          28\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

LAPLACE SMOOTHING

```
def laplace_prob(count, total, vocab):
    return (count + 1) / (total + vocab)
```

SENTENCE PROBABILITY

```
def sentence_probability(sentence, n):
    tokens = preprocess_text(sentence)
    prob = 1.0

    if n == 1:
        total = len(train_tokens)
        for w in tokens:
            prob *= laplace_prob(uni_count[(w,)], total, vocab_size)

    elif n == 2:
        for i in range(len(tokens) - 1):
            prob *= laplace_prob(
                bi_count[(tokens[i], tokens[i+1])],
                uni_count[(tokens[i],)],
                vocab_size
```

```
            )

    else:
        for i in range(len(tokens) - 2):
            prob *= laplace_prob(
                tri_count[(tokens[i], tokens[i+1], tokens[i+2])],
                bi_count[(tokens[i], tokens[i+1])],
                vocab_size
            )

    return prob
```

SENTENCE PROBABILITY TESTING

```
sentences = [
    "language models are important",
    "artificial intelligence improves education",
    "technology impacts society",
    "students learn using ai tools",
    "language processing is useful"
]

for s in sentences:
    print("\nSentence:", s)
    print("Unigram :", sentence_probability(s, 1))
    print("Bigram  :", sentence_probability(s, 2))
    print("Trigram :", sentence_probability(s, 3))
```

```
Sentence: language models are important
Unigram : 4.5931889415440667e-07
Bigram  : 0.01330712825360077
Trigram : 0.002306529865584984

Sentence: artificial intelligence improves education
Unigram : 6.0614975940264816e-09
Bigram  : 4.9830722105787684e-05
Trigram : 9.086778736937755e-05

Sentence: technology impacts society
Unigram : 2.025682023148855e-06
Bigram  : 0.06688936610196453
Trigram : 0.29292929292929293

Sentence: students learn using ai tools
Unigram : 9.494204993735008e-13
Bigram  : 1.2299171592987108e-08
Trigram : 2.7939906848350567e-06

Sentence: language processing is useful
```

```
Unigram : 2.749277178633188e-09
Bigram  : 0.000365515661759369
Trigram : 0.002958881746760535
```

PERPLEXITY FUNCTION

```
def perplexity(sentence, n):
    tokens = preprocess_text(sentence)
    N = len(tokens)
    prob = sentence_probability(sentence, n)
    return pow(1/prob, 1/N)
```

PERPLEXITY COMPARISON

```
for s in sentences:
    print("\nSentence:", s)
    print("Unigram Perplexity:", perplexity(s, 1))
    print("Bigram Perplexity :", perplexity(s, 2))
    print("Trigram Perplexity:", perplexity(s, 3))
```

```
Sentence: language models are important
Unigram Perplexity: 38.41239820495564
Bigram Perplexity : 2.9442786811489436
Trigram Perplexity: 4.563102553237777

Sentence: artificial intelligence improves education
Unigram Perplexity: 113.33264267929046
Bigram Perplexity : 11.902157826026246
Trigram Perplexity: 10.242300458083681

Sentence: technology impacts society
Unigram Perplexity: 79.03320159399013
Bigram Perplexity : 2.4634720538980273
Trigram Perplexity: 1.505725246337524

Sentence: students learn using ai tools
Unigram Perplexity: 253.8097370055186
Bigram Perplexity : 38.196610656250776
Trigram Perplexity: 12.90491086617243

Sentence: language processing is useful
Unigram Perplexity: 138.10053728910415
Bigram Perplexity : 7.232251202643337
Trigram Perplexity: 4.2876378671161275
```

ANALYSIS & OBSERVATIONS

Comparison and Analysis

Bigram and Trigram models generally show lower perplexity than Unigram models. Trigram models capture more context but may suffer from data sparsity. Bigram models often provide the best balance between performance and simplicity. Unseen words receive non-zero probability due to Laplace smoothing. Lower perplexity indicates a better-performing language model.