NLP - EXP - 4

Atharva Prashant Pawar (9427) - [Batch - D]

Test2 : Implement N-gram model for sentiment analysis and analyze the effect of different value of N on the model. prediction

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.1)
```

```
import nltk
from nltk.corpus import movie_reviews
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
from nltk.probability import FreqDist, ConditionalFreqDist
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy

# movie reviews dataset
nltk.download('movie_reviews')
positive_reviews = [(list(movie_reviews.words(fileid)), 'positive') for fileid in movie_reviews.fileids('pos')]
negative_reviews = [(list(movie_reviews.words(fileid)), 'negative') for fileid in movie_reviews.fileids('neg')]
all_reviews = positive_reviews + negative_reviews
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
```

```
# Define the N for N-grams
# N = 2
```

```
def preproccess(N):
  all_ngrams = [ngrams(review, N) for review, _ in all_reviews]  # Createing N-grams for all reviews

  flat_ngrams = [ng for ngram_list in all_ngrams for ng in ngram_list] # Flatten N-grams

  ngram_freq_dist = FreqDist(flat_ngrams)  # Frequency distribution of N-grams

  # Conditional frequency distribution of N-grams based on sentiment
  cfd = ConditionalFreqDist([(ng, sentiment) for ng, sentiment in zip(flat_ngrams, [sent for _, sent in all_reviews])])

  # function to extract features from a review
  def extract_features(review):
      features = {}
[illegible]rue

  featuresets = [(extract_features(review), sentiment) for review, sentiment in all_reviews] # Createing feature sets

  train_set, test_set = featuresets[:1600], featuresets[1600:] # Split train and test sets

  classifier = NaiveBayesClassifier.train(train_set) # Train Naive Bayes classifier

  # Test the classifier
  accuracy_score = accuracy(classifier, test_set)
  print("Accuracy:", accuracy_score)
  return { "N" : N, "Accuracy" : accuracy_score}
```

```
Nval = 5
result = []
for nItem in range(Nval):
  outResult = preproccess(nItem)
  result.append(outResult)
```

```
Accuracy: 0.0
Accuracy: 0.4525
Accuracy: 0.73
Accuracy: 0.89
Accuracy: 0.915
```

```
print(result)
```

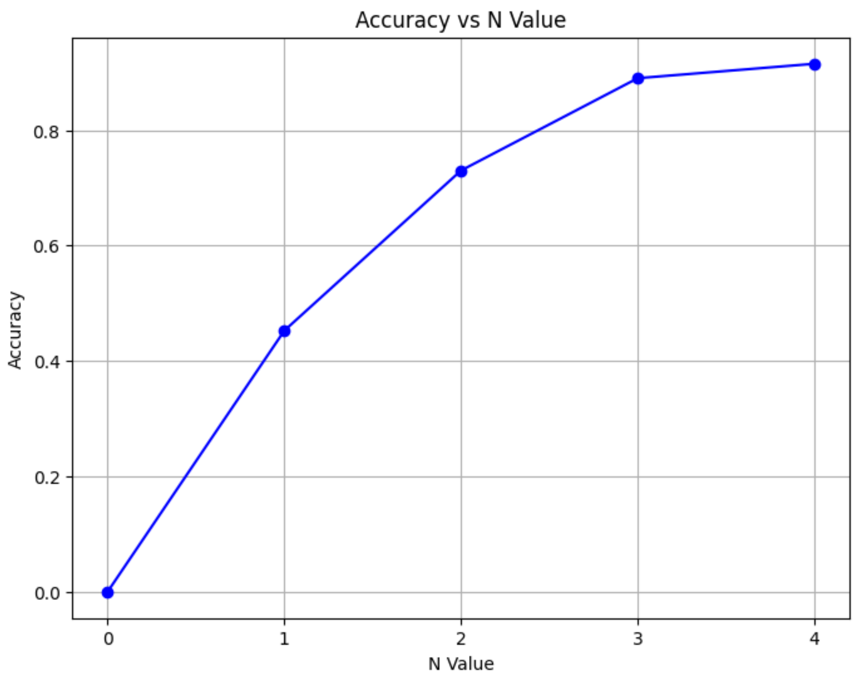
```
[{'N': 0, 'Accuracy': 0.0}, {'N': 1, 'Accuracy': 0.4525}, {'N': 2, 'Accuracy': 0.73}, {'N': 3, 'Accuracy': 0.89}, {'N': 4, 'Accurac
```

```
import matplotlib.pyplot as plt

# [{'N': 0, 'Accuracy': 0.0}, {'N': 1, 'Accuracy': 0.4525}, {'N': 2, 'Accuracy': 0.73}, {'N': 3, 'Accuracy': 0.89}, {'N': 4, 'Accuracy':

# Extract N values and Accuracy values from the result list
n_values = [item['N'] for item in result]
accuracy_values = [item['Accuracy'] for item in result]

plt.figure(figsize=(8, 6))
plt.plot(n_values, accuracy_values, marker='o', linestyle='-', color='b')
plt.title('Accuracy vs N Value')
plt.xlabel('N Value')
plt.ylabel('Accuracy')
plt.xticks(n_values)  # Set x-axis ticks to match N values
plt.grid(True)
plt.show()
```



Saving...

✓ 0s completed at 10:17 AM

N-Grams

Corpus A

Select Corpus

(eos) Can I sit near you (eos) You can sit (eos) Sit near him (eos) I can sit you (eos)

Find Bigram Probabilities

	(eos)	I	you	him	can	near	sit
(eos)	0	0.2	0.2	0	0.2	0	0.2
I	0	0	0	0	0.5	0	0.5
you	0.66	0	0	0	0.33	0	0
him	1	0	0	0	0	0	0
can	0	0.33	0	0	0	0	0.66
near	0	0	0.5	0.5	0	0	0
sit	0.25	0	0.25	0	0	0.5	0

Submit

Find probabilities of the following sentences:

Sentence	Probability
I sit you EOS	0.0826
Can you sit near I EOS	0
I can sit EOS	0.0823
You sit EOS	0

Submit

Wrong Answer

Atharva Prashant Pawar (9427) - [Batch-D]

NLP : Exp-4

Q1. What is perplexity? How do you use perplexity to measure N-gram model's performance?

=> Perplexity is a measure used to evaluate the performance of language models, including N-gram models. It quantifies how well a language model predicts a given sequence of words. A lower perplexity indicates better performance, as it reflects the model's ability to accurately predict the next word in a sequence.

* To calculate perplexity for an N-gram model, follow these steps:

Divide the text data into a training set & a test set. Train the N-gram model on the training set to learn the probabilities of word sequences.

A. Calculate Probabilities:
For a test dataset, calculate the conditional probabilities of each word given its prescending N-1 words using the N-gram model.

B. Compute Perplexity:
Perplexity is calculated as the inverse geometric mean of the probabilities. For a test dataset with words $w_1, w_2, \ldots, w_N$, the perplexity PP is calculated as:

$$PP = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots w_N)}}$$

Where N is the no. of words in the dataset.

(2)

C. Interpretation:
lower perplexity values indicates that the model assigns higher probabilities to the observed sequences of words in the test data, suggesting a better match b/w the model's predictions & the actual data distribution.

~~D. 4~~

D. Model Comparison:
Compare perplexity scores accross different N-gram models or other language models. A lower perplexity value typically indicates a better-performing model, as it suggests the model better captures the underlying language patterns.

E. Tuning & Evaluation:
Perplexity can guide hyperparameters tuning, such as selecting the optimal value of N for N-gram models. It's also used to evaluate models during development to choose the best-performance one.

F. Generalization:
A model with low perplexity is likely to generate well to unseen data, indicating that it captures broader language patterns beyond the training data.