Natural Language Processing

Assignment -1

Gangaram Arvind Sudewad
20CS30017

TASK 1:

Data Preparation:
Download the Treebank corpus from NLTK using nltk.download('treebank'). Preprocess the corpus to extract sentences and their corresponding POS tags.
Build a Transition Probability Matrix:
Calculate the transition probabilities between POS tags based on the training data. This matrix will represent the probability of transitioning from one tag to another in a sentence.
Build an Emission Probability Matrix:
Calculate the emission probabilities of words given their POS tags. This matrix will represent the probability of observing a word for a particular POS tag.
Viterbi Algorithm:
Start with the first word, initializing the matrix with transition probabilities from start tags to initial POS tags. Iterate through the remaining words, calculating probabilities of reaching each tag based on prior probabilities and transition/emission probabilities. Store the maximum probability and its corresponding backpointer for each word.
Initialize Viterbi Matrix:
Create a Viterbi matrix to store the probabilities of possible tag sequences for each word in a sentence.
Initialization:
Start with the first word in the sentence and initialize the Viterbi matrix with the probabilities of transitioning from a start tag to the initial POS tags for the first word.
Recursion:
Iterate through the remaining words in the sentence. For each word, calculate the probability of reaching each POS tag based on the previous word's probabilities and the transition and emission probabilities. Store the maximum probability and the corresponding backpointer (the previous POS tag) in the Viterbi matrix.
Termination:
After processing all words in the sentence, identify the most likely POS tag sequence by backtracking through the Viterbi matrix, starting from the end.
Tagging New Sentences:
Use the trained model to tag new sentences by applying the Viterbi Algorithm to calculate the most likely POS tag sequence for each word in the sentence.
Evaluation:
Evaluate the performance of your POS tagger on a validation or test dataset by comparing the predicted tags to the true tags.

Improvement:
Refine your model by considering more sophisticated features, handling unknown words, and addressing tagging ambiguities.

Accuracy on Test Data: **87.85%**

First few sentences of the dataset:

Sentence 1: Pierre Vinken , 61 years old , will join the board as a non executive director Nov. 29 .
Sentence 2: Mr. Vinken is chairman of Elsevier N.V. , the Dutch publishing group .
Sentence 3: Rudolph Agnew , 55 years old and former chairman of Consolidated Gold Fields PLC , was named *-1 a non executive director of this British industrial conglomerate .
Sentence 4: A form of asbestos once used * * to make Kent cigarette filters has caused a high percentage of cancer deaths among a group of workers exposed * to it more than 30 years ago , researchers reported 0 *T*-1.
Sentence 5: The asbestos fiber , crocidolite , is unusually resilient once it enters the lungs , with even brief exposures to it causing symptoms that *T*-1 show up decades later , researchers said 0 *T*-2 .

First few tagged sentences of the dataset:

Tagged Sentence 1: [('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'), ('old', 'JJ'), (',', ','), ('will', 'MD'), ('join', 'VB'), ('the', 'DT'), ('board', 'NN'), ('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'JJ'), ('director', 'NN'), ('Nov.', 'NNP'), ('29', 'CD'), ('.', '.')]

Tagged Sentence 2: [('Mr.', 'NNP'), ('Vinken', 'NNP'), ('is', 'VBZ'), ('chairman', 'NN'), ('of', 'IN'), ('Elsevier', 'NNP'), ('N.V.', 'NNP'), (',', ','), ('the', 'DT'), ('Dutch', 'NNP'), ('publishing', 'VBG'), ('group', 'NN'), ('.', '.')]

Tagged Sentence 3: [('Rudolph', 'NNP'), ('Agnew', 'NNP'), (',', ','), ('55', 'CD'), ('years', 'NNS'), ('old', 'JJ'), ('and', 'CC'), ('former', 'JJ'), ('chairman', 'NN'), ('of', 'IN'), ('Consolidated', 'NNP'), ('Gold', 'NNP'), ('Fields', 'NNP'), ('PLC', 'NNP'), (',', ','), ('was', 'VBD'), ('named', 'VBN'), ('*-1', '-NONE-'), ('a', 'DT'), ('nonexecutive', 'JJ'), ('director', 'NN'), ('of', 'IN'), ('this', 'DT'), ('British', 'JJ'), ('industrial', 'JJ'), ('conglomerate', 'NN'), ('.', '.')]

Tagged Sentence 4: [('A', 'DT'), ('form', 'NN'), ('of', 'IN'), ('asbestos', 'NN'), ('once', 'RB'), ('used', 'VBN'), ('*', '-NONE-'), ('*', '-NONE-'), ('to', 'TO'), ('make', 'VB'), ('Kent', 'NNP'), ('cigarette', 'NN'), ('filters', 'NNS'), ('has', 'VBZ'), ('caused', 'VBN'), ('a', 'DT'), ('high', 'JJ'), ('percentage', 'NN'), ('of', 'IN'), ('cancer', 'NN'), ('deaths', 'NNS'), ('among', 'IN'), ('a', 'DT'), ('group', 'NN'), ('of', 'IN'), ('workers', 'NNS'), ('exposed', 'VBN'), ('*', '-NONE-'), ('to', 'TO'), ('it', 'PRP'), ('more', 'RBR'), ('than', 'IN'), ('30', 'CD'), ('years', 'NNS'), ('ago', 'IN'), (',', ','), ('researchers', 'NNS'), ('reported', 'VBD'), ('0', '-NONE-'), ('*T*-1', '-NONE-'), ('.', '.')]

Tagged Sentence 5: [('The', 'DT'), ('asbestos', 'NN'), ('fiber', 'NN'), (',', ','), ('crocidolite', 'NN'), (',', ','), ('is', 'VBZ'), ('unusually', 'RB'), ('resilient', 'JJ'), ('once', 'IN'), ('it', 'PRP'), ('enters', 'VBZ'), ('the', 'DT'), ('lungs', 'NNS'), (',', ','), ('with', 'IN'), ('even', 'RB'), ('brief', 'JJ'), ('exposures', 'NNS'), ('to', 'TO'), ('it', 'PRP'), ('causing', 'VBG'), ('symptoms', 'NNS'), ('that', 'WDT'), ('*T*-1', '-NONE-'), ('show', 'VBP'), ('up', 'RP'), ('decades', 'NNS'), ('later', 'JJ'), (',', ','), ('researchers', 'NNS'), ('said', 'VBD'), ('0', '-NONE-'), ('*T*-2', '-NONE-'), ('.', '.')]

```
test_sentence = "My name is Gangaram Sudewad ."
```

```
Word: My,          Predicted Tag: DT
Word: name,        Predicted Tag: NN
Word: is,          Predicted Tag: VBZ
Word: Gangaram,    Predicted Tag: VBN
Word: Sudewad,     Predicted Tag: -NONE-
Word: .,           Predicted Tag: .
```

TASK 2:

Sample words from the dataset: ['plot', ':', 'two', 'teen', 'couples', 'go', 'to', 'a', 'church', 'party', ',', 'drink', 'and', 'then', 'drive', '.', 'they', 'get', 'into', 'an']

The provided Python code performs sentiment analysis on the movie_reviews dataset using a Multinomial Naive Bayes classifier.

Data Preparation:
The movie_reviews dataset is loaded from NLTK's corpus collection.
The documents (movie reviews) are labeled as positive or negative and shuffled randomly.
Feature Extraction:
TF-IDF (Term Frequency-Inverse Document Frequency) is used to convert the textual data into numerical features.
The TF-IDF vectorizer is limited to 5000 features.
Data Splitting:
The data is split into training, validation, and test sets.
80% of the data is used for training, 10% for validation, and 10% for testing.
Model Training:
A Multinomial Naive Bayes classifier is trained on the training data.
Validation accuracy is calculated to assess the model's performance on unseen data.
Final Model Training:
The classifier is retrained on the combined training and validation data to make use of all available labeled data.
Model Testing:
The final model is used to make predictions on the test set.
Test accuracy is calculated to evaluate the model's performance.
Classification Report:
A classification report is generated, providing precision, recall, F1-score, and support for each class (positive and negative).
Results Printing:
The code prints the validation accuracy, test accuracy, and the classification report.

The Multinomial Naive Bayes classifier achieves a certain accuracy level in classifying movie reviews as positive or negative. A classification report further details the model's performance in terms of precision, recall, and F1-score for each class. This code serves as a foundation for sentiment analysis and can be extended or optimized for more advanced natural language processing tasks.

Results:

Validation Accuracy: 0.84
Test Accuracy: 0.99

	precision	recall	f1-score	support
neg	1.000000	0.989583	0.994764	96.000
pos	0.990476	1.000000	0.995215	104.000
accuracy	0.995000	0.995000	0.995000	0.995
macro avg	0.995238	0.994792	0.994990	200.000
weighted avg	0.995048	0.995000	0.994999	200.000

TASK 3:

performs text classification on movie reviews using a combination of TF-IDF (Term Frequency-Inverse Document Frequency) features and POS (Part-of-Speech) tag features. It uses the scikit-learn library for machine learning and NLTK (Natural Language Toolkit) for text processing. Here's an explanation of each part of the code:

Importing Libraries:
The code starts by importing the necessary Python libraries, including nltk, sklearn, pandas, and others.
Downloading NLTK Data:
It downloads NLTK data for tokenization and POS tagging using nltk.download() for the 'punkt', 'averaged_perceptron_tagger', and 'movie_reviews' datasets.
Loading and Shuffling Data:
The movie reviews dataset is loaded from NLTK's movie_reviews corpus. It contains positive and negative movie reviews, which are shuffled to ensure randomness.
Feature Extraction with TF-IDF:
TF-IDF vectorization is performed on the movie review texts. The TfidfVectorizer is initialized with a maximum of 2000 features. It converts the text data into a numerical format, representing the importance of each word in the reviews.
Splitting Data:
The data is split into training, validation, and test sets. The train_test_split function is used twice to perform this split. Initially, it's split into 80% training and 20% temporary data, and then the temporary data is further split into validation (50%) and test (50%) sets.
Extracting POS Tags:
A function extract_pos_tags is defined to extract POS tags from each document. It tokenizes the text and extracts the POS tags using NLTK's pos_tag function.
Combining Features:
TF-IDF features are converted back to strings, and then POS tag features are extracted using the extract_pos_tags function. The TF-IDF and POS tag features are combined into a single feature vector for each document.

TF-IDF Vectorization for Combined Features:
A new TfidfVectorizer is initialized for the combined features, and TF-IDF vectorization is performed again. This time, the vectorizer is initialized with a maximum of 3000 features.
Training the Classifier:
A Multinomial Naive Bayes classifier (MultinomialNB) is trained on the combined TF-IDF and POS tag features using the training data. Validation and Test Predictions: The trained model is used to make predictions on the validation and test sets.
Calculating Accuracy:
The accuracy of the model is calculated for both the validation and test sets using scikit-learn's accuracy_score.
Classification Report:
A classification report is generated for the test set, including metrics such as precision, recall, F1-score, and support for each class. This information is stored in a pandas DataFrame for easy analysis.
Print Results:
Finally, the code prints the validation and test accuracy with combined features, along with the classification report for the test set.

Observation:
Validation Accuracy with Combined Features: 0.45
Test Accuracy with Combined Features: 0.51

	precision	recall	f1-score	support
neg	0.5100	1.00	0.675497	51.00
pos	0.0000	0.00	0.000000	49.00
accuracy	0.5100	0.51	0.510000	0.51
macro avg	0.2550	0.50	0.337748	100.00
weighted avg	0.2601	0.51	0.344503	100.00

Summary:
Taken steps to combine information from TF-IDF word frequencies and POS tags to create a more informative feature set for sentiment analysis. This approach can be beneficial as it considers both lexical and grammatical aspects of the text.