Homework 2

CS/CE 458/463 Natural Language Processing
Habib University

Fall 2024

Instructor: Dr. Ayehsa Enayat
Abeeha Zehra – az07728

Q1: Implementing an N-Gram Language Model and Testing Perplexity

a. Approach to implementing the N-gram model.

Answer

In this task, I began by processing the data from the training dataset, where I performed data cleaning and constructed a vocabulary. To train the model on n-grams, I first segmented the entire text into n pairs, where n indicates whether it's a unigram, bigram, trigram, etc. I then calculated the frequency of each n-gram and stored these frequencies in a dictionary, while also creating a separate dictionary for the n-1 grams.

Next, I calculated the probabilities for the n-grams, applying Laplace smoothing to account for potential unseen words in the test data. This smoothing was necessary to ensure more accurate probability calculations using the formula given in equation 1.

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{c(w_{n-1}) + V} \tag{1}$$

After storing the probabilities and training the model, I processed the test data. I then utilized the previously calculated probabilities to predict the likelihood of words in the test set. If the sequence was present in the n-gram model, its corresponding probability was used; if not found, I checked the n-1 gram. I applied smoothing and stored the resulting value if that also yielded no results. Finally, these probabilities were used to compute perplexity using the formula in equation 2.

$$PP(W) = \sqrt{N \prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}} \tag{2}$$

b. Discuss the results of your perplexity calculations for different ‘n‘ values.

Answer

According to the code, the perplexity value for different n-grams are:

N-gram	Perplexity
Unigram	30.79698644848602
Bigram	22.482921683412396
Trigram	26.813564348514195
Quadgram	29.471725107905655
Pentagram	30.036455268318882

Upon analyzing the perplexity results for different n-gram models, it is evident that the unigram model has the highest perplexity. This can be attributed to the fact that unigrams only account for individual words without considering any surrounding context, which leads to poor prediction performance. Additionally, since many features from the test set were likely absent from the training data, this lack of context contributed to the higher perplexity.

Conversely, the bigram model yields the lowest perplexity, indicating that using the previous word to predict the next provides meaningful context, enhancing prediction accuracy. However, as the sequence size increases beyond bigrams, there is a noticeable rise in perplexity for trigrams, quadgrams, and higher-order models. This can be explained by data sparsity—the training set likely does not contain enough instances of longer n-grams, causing a higher number of unseen n-grams in the test set, leading to reduced performance.

From these results, we can conclude that the bigram model performs best for this dataset. It balances context and accuracy, while higher n-grams suffer from increased perplexity due to data sparsity and fewer examples in the training set.

Q2: Implementing a Naive Bayes Classifier for Sentiment Analysis

a. Approach of implementing the Naive Bayes Classifier and its performance.

Answer

To implement the Naive Bayes classifier, I first trained the model using the training set. The initial step involved preprocessing the data by removing punctuation, special characters, and stop words. Next, the text was tokenized by splitting it based on spaces. After this preprocessing, a vocabulary was created, and word counts were computed for each class. The probability of each word in each class was then calculated. Once the model was trained, I processed 100 files from the test data by forming tokens. For each review in the test data, I discarded words that were not present in the training set, and for the remaining words, I used the probabilities previously calculated during training. After determining the probabilities for both the positive and negative classes, I compared them, and the class with the higher probability was selected as the prediction and for that I used the formula given in equation 3.

$$\hat{C} = \arg\max_{C_k} \left[P(C_k) \cdot \prod_{i=1}^{n} P(X_i|C_k) \right] \tag{3}$$

To evaluate the accuracy of the predictions, I compared them with the actual outcomes and constructed a confusion matrix, which recorded the following results:

Metric	**Value**
True Positive	44
True Negative	92
False Positive	8
False Negative	56

On the basis of this confusion matrix, I calculated accuracy, precision, recall, and f1_score and these are shared in the table below:

Evaluation	**Score**
Accuracy	0.68
Precision	0.84
Recall	0.44
F1_score	0.57

The Naive Bayes classifier shows moderate accuracy (68%) and high precision (84.6%), meaning it's good at making correct positive predictions. However, its low recall (44%) indicates it misses many true positives. The F1 score (57.9%) reflects an overall moderate balance, but the model struggles to capture all relevant positive cases. Overall, it performs well in avoiding false positives but poorly in identifying all positive instances.

b. How to improve the performance of the model.

Answer

After measuring the performing of my model, it can be seen that the model is struggling to identify mostly positive instances. Hence, to improve the performance what we can do is:

- Try lemmatization or stemming to normalize words and reduce the dimensionality of the vocabulary.
- Instead of Laplace smoothing, we can try k-smoothing.

c. Any challenges you encountered while implementing the model.

Answer

Implementing the model was relatively straightforward, as we had already discussed the Naive Bayes calculations in class. Using the provided formulas and my understanding of the model, I was able to implement it successfully. However, verifying the results was time-consuming due to the large size of the input training and test files.

Q3: Implementing an Artificial Neural Network for Sentiment Classification

a. Summarize the performance and the results generated by the ANN model.

Answer

For the development of the artificial neural network (ANN), I applied the foundational concepts learned in class. This involved implementing forward propagation which I did using equation 4, initializing weights and biases, and utilizing backpropagation to compute errors and update the weights and biases which I did using equations (5-9).

$$y = \frac{1}{1 + e^{-(\sum_{i=1}^{n} w_i x_i + b)}} \tag{4}$$

$$Err_k = o_k(1 - o_k)(t_k - o_k) \tag{5}$$

$$Err_j = o_j(1 - o_j) \sum_k (Err_k W_{jk}) \tag{6}$$

$$W_{ij} = W_{ij} + \eta \cdot Err_j \cdot o_i \tag{7}$$

$$b_k = b_k + \eta \cdot Err_k \tag{8}$$

$$b_j = b_j + \eta \cdot Err_j \tag{9}$$

This process was repeated over 10 epochs, after which the final weights and biases were used to predict the output for the test dataset. Based on the model's predictions, the following confusion matrix was generated:

Metric	Value
True Positive	73
True Negative	188
False Positive	61
False Negative	178

Based on the above confusion matrix, I calculated accuracy, precision, recall, and f1 score and these are shared in the table below:

Evaluation	Score
Accuracy	0.52
Precision	0.54
Recall	0.29
F1_score	0.37

Answer

The ANN model achieved an accuracy of 52.2%, meaning it correctly predicted the class labels just over half the time. The precision of the model was 54.5%, indicating that when it predicted a positive class, it was correct slightly more than half the time. However, the model's recall was relatively low at 29.1%, which shows that it failed to identify a significant number of true positive cases, leading to a high number of false negatives. The F1 score, a metric that balances precision and recall, was 37.9%. This relatively low F1 score suggests that the model's overall ability to correctly classify instances, especially in balancing precision with recall, was poor. However, these results were based on a single run of the model. When the model was run multiple times, the scores varied, as the weights and biases are initialized randomly for each run, leading to different outcomes. Over several runs, the accuracy fluctuated between 49% and 53%, further indicating that the model's performance was inconsistent and generally suboptimal.

b. How to improve the performance of the model

Answer

From the above discussion, we can conclude that the performance of the model was not good. Hence, to improve the performance what we can do is:

- Data Preprocessing: Normalize or standardize input features to enhance convergence.
- Model Architecture: Increase the number of layers or neurons to capture more complex patterns.
- Activation function: We can use different activation function such as ReLU, to improve model performance and enhance its ability to learn complex patterns.
- Hyperparameter: We can change the learning rate parameter for training of the model.

c. Any challenges you encountered while implementing the model

Answer

While data processing, forward propagation, and the initialization of weights and biases were relatively straightforward tasks, I encountered some difficulties with the implementation of backpropagation. This aspect of the process required a deeper understanding and attention to detail, making it more challenging to execute effectively. Furthermore, I faced confusion regarding the appropriate methods for testing the dataset, which added to the complexity of the task and resulted in a more time-consuming experience overall.