Natural Language Processing

Assignment 2

Fall 2024

Instructor: Ayesha Enayet

Q1: Assignment: Implementing an N-Gram Language Model and Testing Perplexity [20]

Objective:

In this assignment, you will implement an N-Gram language model in Python using a text corpus provided in a `train.txt` file. After training your N-Gram model, you will evaluate its performance by calculating the Perplexity on a given test set (test.txt).

Instructions:

1. Load and Preprocess Data:

You are given a `train.txt` file that contains a corpus of text. Load the text file, preprocess it by converting everything to lowercase, removing special characters, and tokenizing the text. Data Link: https://drive.google.com/drive/folders/15f6FQJF-RVzkaGZlzj5Vqqb2xAoxsjKI?usp=drive_link

2. Implement N-Gram Mode:

Implement an N-Gram model with a configurable `n` value (for example, bigram or trigram). Your implementation should:

- Create a vocabulary based on the training data.
- Calculate the probability of each word given the previous `n-1` words.
- Use smoothing techniques (e.g., add-one smoothing) to handle unseen words.

3. Perplexity Calculation:

Implement a function to calculate the perplexity of your model on the test set.

4. Test Different N-Grams:
   - Implement the model for different values of `n` (e.g., unigram, bigram, trigram).
   - Compare the perplexity of the different models on the same test set.

Deliverables:

1. Python Code: [10 points]

A Python script implementing the N-Gram model and perplexity calculation.

The code should include:

- Functions to load and preprocess the data.
- Functions to generate N-Grams and calculate probabilities.
- A function to compute the perplexity on the test set.

2. Report: [10 points]

A brief report explaining:

- Your approach to implementing the N-Gram model.
- The results of your perplexity calculations for different `n` values.
- Discussion of how `n` impacts the performance of the model and perplexity.

Q2: Implementing a Naive Bayes Classifier for Sentiment Analysis [40]

Objective:

In this assignment, you will implement a Naive Bayes classifier from scratch, without using any external libraries (such as scikit-learn) for model training or prediction. You will train your model on the subset of IMDB Movie Reviews Dataset and evaluate its performance on the test data.

Instructions:

You are tasked with implementing a Naive Bayes classifier to perform binary sentiment classification (positive/negative) on the IMDB dataset. Follow the steps below to complete the assignment.

Step 1: Dataset Preparation

1. Download the IMDB Dataset:
   - Use the [IMDB Dataset](https://ai.stanford.edu/~amaas/data/sentiment/) containing 50,000 movie reviews.
   - Implement a code to **select**, **read**, and **load** 500 **positive** and 500 **negative** samples from the **training set**.
   - Implement a code to **select**, **read**, and **load** 100 **positive** and 100 **negative** samples for the testing from the test set.
2. Preprocess the Dataset:
   - Text Cleaning: Remove stop words, punctuations, and convert all text to lowercase.
   - Tokenization: Split each review into individual words (tokens).
   - Vocabulary Building: Build a vocabulary of unique words from the training data.
   - Feature Extraction: For each review, calculate the frequency of each word from the vocabulary. These frequencies will serve as the features for the classifier.

Step 2: Naive Bayes Classifier Implementation

You are required to implement a Naive Bayes classifier using the following steps:

1. Prior Probability Calculation:
   - Calculate the prior probabilities for each class (positive and negative reviews).
   - This can be done by counting how many reviews belong to each class in the training data.
2. Likelihood Estimation:
   - For each word in the vocabulary, calculate the likelihood of that word appearing in a positive or negative review.
   - Use Laplace smoothing to handle words that may not appear in the training data.
3. Naive Bayes Classifier:

- Use the Naive Bayes formula to calculate the posterior probability for each class (positive and negative) for each test review.
- Assign the review to the class with the highest posterior probability.

Step 3: Training and Testing the Classifier

1. Training:
   - Train your Naive Bayes classifier on the training set of 1000 reviews.
   - Use the likelihoods and priors calculated from the training data to build the model.
2. Testing:
   - Test your classifier on the 500 reviews in the test set.
   - For each test review, compute the posterior probabilities for both positive and negative classes and assign the label with the highest probability.

Step 4: Evaluation

1. Accuracy:
   - Calculate the accuracy of your model on the test set (percentage of correctly classified reviews).

2. Confusion Matrix:
   - Generate a confusion matrix to evaluate how well the classifier distinguishes between positive and negative reviews.
3. Precision, Recall, F1-Score:
   - Compute precision, recall, and F1-score for both positive and negative classes.

Step5: Report:

Write a brief report describing:

- The overall performance of the classifier (F1-score, confusion matrix).
- Any challenges you encountered while implementing the model.
- Ideas for improving the model (e.g., using n-grams, improving text preprocessing).

<u>Deliverables:</u>

1. Python Code: A well-documented Python script that includes:
   - Data loading, text preprocessing, tokenization, and feature extraction.
   - Implementation of Naive Bayes from scratch.
   - Training and testing on the IMDB dataset.
   - Evaluation metrics (accuracy, confusion matrix, precision, recall, F1-score).
2. Report: A concise report summarizing the results and your experience with implementing Naive Bayes.

<u>Submission:</u>

- Submit the Python script and the report as a .zip file.
- Ensure the code is well-commented and readable.

<u>Grading Criteria:</u>

- **Correctness** of Naive Bayes implementation: 15 points

- **Performance** on the test set (accuracy, confusion matrix, etc.):5 points
- **Code quality** (documentation, efficiency, readability): 10 points
- **Report** (clarity, insights, evaluation metrics): 10 points

Q3: Implementing an Artificial Neural Network for Sentiment Classification

[40 points]

Objective:

In this assignment, you will implement an Artificial Neural Network (ANN) from scratch, without using machine learning libraries (like TensorFlow or PyTorch) for training and prediction. You will train your ANN on a sentiment classification task using **one-hot encoded** vectors as input.

Instruction:

Step 1: Dataset Preparation

1. Download the Dataset:
   - Use the synthetic Dataset , containing 5000 training and 500 test samples.
     Data Link: (https://drive.google.com/drive/folders/15f6FQJF-RVzkaGZlzj5Vqqb2xAoxsjKI?usp=drive_link)
2. Text Preprocessing:
   - Text Cleaning: Remove punctuations, special characters, and convert all text to lowercase.
   - Tokenization: Tokenize the samples into individual words.
   - Vocabulary Building: Create a vocabulary containing the unique words across the entire dataset. Each word in the vocabulary will be assigned a unique index.
3. One-Hot Encoding:
   - For each sample, convert the tokenized words into one-hot encoded vectors.
   - The length of each vector will be the size of the vocabulary, with `1` indicating the presence of the word in the review and `0` indicating its absence.

Step 2: Implementing an Artificial Neural Network (ANN)

You are required to implement a basic Artificial Neural Network with the following structure:

1. Input Layer:
   - The input to the network will be the one-hot encoded vectors representing each sample.
   - The size of the input layer will be equal to the size of the vocabulary (number of unique words in the dataset).
2. Hidden Layer:
   - Implement at least one fully connected hidden layer. You can choose the number of neurons in the hidden layer (recommended: 128 or 256 neurons).
   - Use sigmoid or ReLU activation function for the hidden layer.
3. Output Layer:
   - The output layer will consist of a single neuron, with a sigmoid activation function to predict the sentiment of the review (0 for negative, 1 for positive).
4. Backpropagation:

- Implement the backpropagation algorithm to update parameters.

6. Training:

- Train the network using the training set(5,000 samples) for a fixed number of epochs (e.g., 10 epochs).
- Update the weights after each epoch based on the training data.

Step 3: Training and Testing

1. Training:
    - Train your neural network on the training set, using one-hot encoded vectors as input.
2. Testing:
    - After training, test the performance of the neural network on the test set (500 samples).
    - For each review, use the trained ANN to predict its sentiment (positive/negative).

Step 4: Evaluation

1. Accuracy:
    - Calculate the accuracy of your model on the test set (percentage of correctly classified reviews).
2. Confusion Matrix:
    - Generate a confusion matrix to visualize the performance of the model in classifying positive and negative reviews.
3. Precision, Recall, F1-Score:
    - Compute precision, recall, and F1-score for both the positive and negative classes.

Step 5: Report:

- Write a brief report summarizing the results:
- The overall performance of the ANN.
- Any challenges you encountered during implementation.
- Any ideas for improving the model (e.g., more hidden layers, different activation functions).

Programming Structure:

You must follow a structured programming approach by implementing the following components as separate functions:

- Initialization: A function to initialize the weights and biases of the neural network.
- Forward: A function to perform the forward propagation through the network.
- Activation Function: Sigmoid and ReLU
- Backward: A function to update the weights and bias using backpropagation.
- Predict: A function to predict the class labels for test set.
- Train: A function to train a model.
- Evaluation: A function to evaluate the performance of the network, including metrics such as accuracy, confusion matrix, precision, recall, and F1-score.

Deliverables:

1. Python Code:

Submit a well-documented Python script that includes:

- Preprocessing of the text data (cleaning, tokenization, one-hot encoding).
- Implementation of the ANN (forward pass, backpropagation, weight updates).
- Training and testing on the synthetic dataset.
- Evaluation metrics (accuracy, confusion matrix, precision, recall, F1-score).

2. Report:
   - A concise report summarizing the results, challenges, and potential improvements.

Grading Criteria:

Correctness of ANN implementation: 15 points

Performance on the test set (accuracy, confusion matrix, etc.): 5 points

Code quality (documentation, efficiency, readability): 10 points

Report (clarity, insights, evaluation metrics): 10 points

Submission:

- Submit the Python script and report as a ".zip" file.
- Ensure the code is well-commented and follows best coding practices.

Good luck!