Sequential Probability Estimation Model with Trigram Approach Report

+ Group E
+ Members:
- Nathaniel Navarro
- Vu Le
- Noah McIvor
- Sean Thean Sea

Introduction

For this assignment, we focus on designing and implementing an advanced probabilistic model that predicts the likelihood of a sequential element within a given context. This task involves constructing a trigram-model to predict the likelihood of a third element based on two predecessors. In addition, the model should be capable of evolving with increasing data. The completion of this assignment will demonstrate proficiency in applying complex statistical concepts and machine learning techniques in natural language processing (NLP). It also shows proficiency in manipulating large data sets, implementing advanced mathematical models, and developing algorithms for accurate analysis of sequential data.

Mathematical Formulations

We used a few Mathematical formulations such as P(z|x,y) = Count(x,y,z)/Count(x,y) to calculate the conditional probability for each trigram (x,y,z). However, there are some challenges of sparse data and potential for zero-probability of occurrences of trigram. To handle these challenges, we used another Mathematical formulation which is P_smooth(z|x,y) = (Count(x,y,z) + ɑ)/(Count(x,y) + ɑ·N). Next, we implemented Bayesian Inference and Maximum Likelihood Estimation (MLE) to optimize the probability distributions: θ = argmax L(θ;x,y,z). We then used P(x_1, x_2,...., x_n) = P(x_1)P(x_2|x_1)P(x3|x_1,x_2)....P(x_n|x_n-2, x_n-1) to calculate the joint probability of a sequence of tokens. Lastly, for Model performance metrics such as precision, recall, and F1-score, we used a few formulas listed below:

+ Precision = (True Positive) / (True Positive + False Positive)
+ Recall = (True Positive) / (True Positive + False Negative)
+ F1-score = (2 * Precision * Recall) / (Precision + Recall)

Implementation

Preprocessing. The list of sentences we have obtained from the Brown Corpus underwent preprocessing steps to clean text data and reduce variability. The steps we have performed are tokenization, normalization, and optional lemmatization. Before performing these steps, we concatenate the elements in the sentence list into a string where each element is separated by a space. During normalization, we lowercased each letter in the string. Afterwards, we performed tokenization, in which we broke down the string into a list of words where each element contains a single word or punctuation. In lemmatization, words are reduced into their base/root form. For example, the word “running” is changed to “run.” We made this step optional using a boolean parameter within the preprocessing function. Finally, we inserted start (<s>) and end (</s>) tokens, respectively, at the beginning and end of each representation of a sentence.

Trigram Construction. To build our trigram model, we initialized a nested dictionary with keys being represented as tuples, containing the first and second word of the trigram, and the values being inner dictionaries in which the keys are the third words in the trigrams and the values as the occurrence count of each trigram. Each entry in the inner dict essentially represents a trigram, and the probability of the last token, given the first two tokens. During construction, we iterate through processed sentences, extracting trigrams with padding on both sides. Laplace smoothing is applied to each trigram, ensuring there are no non-zero probabilities.

Classifier Implementation. The classifier takes in a sequence and splits it into trigrams. Using the trigrams it'll determine which genre that the trigram most likely belongs to. We keep track of the probabilities of each genre and output the maximum one at the end.

Evaluation Methodology

Dataset Splitting. To split the data, we began by defining the proportions of the processed sentences dataset. We then introduced randomness in the dataset by shuffling the order of the processed sentences. Using the total number of sentences in the dataset, we calculated the ratios for the training, validation, and test sets and then used these ratios to split the dataset.

Training Procedure. The classifier was trained using a trigram language model approach. We built the trigram and used Laplace smoothing to handle unseen trigram and zero probabilities. Bayesian inference and MLE were used to update the probabilities and optimize the model parameters.

Model Performance Metrics. To evaluate the classifier’s performance, we used precision, recall and F1-score metrics. All of these measure the proportion of correctness of predicted next words.

Performance on Training and Validation Set. The classifier archived high accuracy on both training and validation sets. The validation was used to assess the classifier’s generalization capability. By monitoring the performance metrics such as precision, F1-score and recall, we gained some insights into the classifier’s effectiveness and potential areas of improvement and mistakes.

Conclusion

In summary, the trigram model effectively understands and predicts sequential language problems. Challenges such as zero probability and sparse data indicate the importance of implementing smoothing techniques such as Laplace smoothing. Constructing the model while utilizing these techniques offers insight on the prediction of language patterns. Future improvements will focus on improving predictive accuracy. This assignment highlights the evolution and refinement needed for NLP models.