Assignment 1: Probabilistic and Nearest Neighbor Methods in Textual Analysis (Report)

Jorge Castro

Department of Computer Sc. and Eng.

University of South Florida, Tampa, Florida, USA

1. Introduction and Scope

This report explores the use of probabilistic models and the K-Nearest Neighbors (KNN) algorithm for textual analysis within the nltk corpus, focusing on classification and feature extraction. It details the execution and analysis of three Python scripts, a trigram model, a Naive Bayes classifier, and a parallel-processed KNN classifier, each highlighting a different aspect of NLP.

Report Structure:

Executing Python Scripts: Offers step-by-step instructions for running the scripts, including environment setup and dependency management.

Analysis of Each Script: Breaks down the functionality, application, and evaluation of the trigram model (part1_manual_trigram.py), Naive Bayes classifier (part2_bayesian.py), and KNN classifier (part3_knn_parallel.py), emphasizing their contributions to understanding and processing language.

Comparative Analysis: Compares these methodologies, assessing their performance and applicability to NLP tasks, responding to the assignment's call for a detailed comparative study.

Math : Here we will look at some of the fundamental math equations needed to understand the models we are building.

Conclusion: Discusses potential enhancements and applications which go beyond the theory explored in this report.

2. Executing Python Scripts

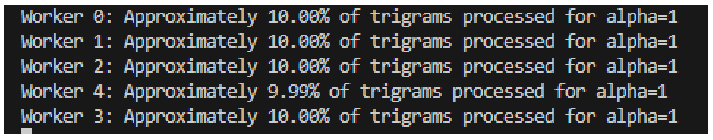
To run the python scripts simply navigate to the source directory and run the scripts in order. Part2 and Part3 reference CSVs from previous runs.
To setup the environment use pip for any missing modules.

```
python part1_manual_trigram.py
python part2_bayesian.py
python part3_knn_parallel.py
```

3. Analysis of Each Script

Trigram Model (part1_manual_trigram.py)

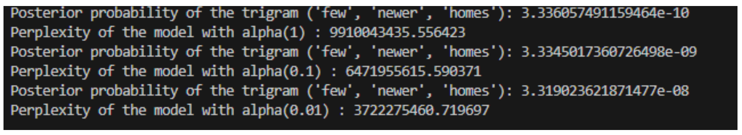
This script is set up to calculate trigram conditional probability given state of bigram. The goal is to build a probabilistic model that predicts the third word given the first two. The calculation of this probability took a very long time, so the script was setup to compute this probability in parallel using all available cores on the user's hardware. If more information is extracted about the users GPU this calculation can be solved x1000 faster. Another version of this calculation was performed but not submitted which instead counted each time a trigram was seen. Additionally, concepts of MLE is exercised by looping through different alpha values.



```
Worker 0: Approximately 10.00% of trigrams processed for alpha=1
Worker 1: Approximately 10.00% of trigrams processed for alpha=1
Worker 2: Approximately 10.00% of trigrams processed for alpha=1
Worker 4: Approximately 9.99% of trigrams processed for alpha=1
Worker 3: Approximately 10.00% of trigrams processed for alpha=1
```

Naive Bayes Classifier (part2_bayesian.py)

This script implements Naive Bayes to classify the text into classes/ feature clusters in an unsupervised manner. Then we calculate the perplexity of the model with different alphas. The perplexity of this classifier performed better when the feature extracted was frequency of trigrams instead of using conditional probabilities of a trigram given a bigram.



```
Posterior probability of the trigram ('few', 'newer', 'homes'): 3.336057491159464e-10
Perplexity of the model with alpha(1) : 9910043435.556423
Posterior probability of the trigram ('few', 'newer', 'homes'): 3.3345017360726498e-09
Perplexity of the model with alpha(0.1) : 6471955615.590371
Posterior probability of the trigram ('few', 'newer', 'homes'): 3.319023621871477e-08
Perplexity of the model with alpha(0.01) : 3722275460.719697
```

KNN Classifier with Parallel Processing (part3_knn_parallel.py)

This script builds and evaluates a KNN model by extracting frequency count / bag of words. The classification / clustering was determined by using Euclidean Distance which is the most common metric. Euclidean distance calculates the straight-line distance between two points in space. It's intuitive and works well for low-dimensional data but can become less reliable in high-dimensional spaces due to the curse of dimensionality.

3. Comparative Analysis

This section evaluates the trigram model, Naive Bayes classifier, and KNN classifier with parallel processing, highlighting their strengths and challenges in NLP tasks.

Trigram vs. Naive Bayes: Trigram models are effective for sequence prediction but struggle with sparse data. Naive Bayes classifiers excel in text classification due to their probabilistic approach, albeit assuming feature independence which may not always hold.

Trigram vs. KNN: While trigrams focus on immediate context, making them suitable for text generation, KNN uses similarity measures for classification, facing challenges with high-dimensional data but benefiting from parallel processing for efficiency.

Naive Bayes vs. KNN: Naive Bayes is computationally efficient for high-dimensional datasets, whereas KNN's adaptability is offset by its computational demands, although mitigated by parallel processing.

4. Math

The core component of KNN is finding neighbors based on distance, understanding the mathematical foundation behind these distance metrics is crucial for effectively implementing and optimizing the classifier. Distance metrics determine how similarities between data points are quantified, directly impacting classification accuracy and model performance.

Euclidean Distance: Calculates straight line distance and can be used in matrix format to expedite the processing time.

Manhattan Distance: Manhattan distance sums the absolute differences of their Cartesian coordinates. It's particularly useful in grid-like data structures and can be more robust than Euclidean in certain high-dimensional scenarios.

Cosine Similarity: Instead of measuring distance, cosine similarity evaluates the cosine of the angle between two vectors, offering a measure of orientation similarity rather than magnitude. This approach is highly effective in text classification, where the orientation of the feature vectors (word counts or TF-IDF scores) can be more informative than their direct distances.

5. Conclusion

In conclusion, this report has meticulously detailed the process for employing the provided scripts, highlighting their application of probabilistic models and distance metrics for feature extraction from the nltk corpus library. Looking ahead, the potential for advancing these methodologies beckons further exploration, particularly in the realms of test data preprocessing and the impact of training with the entirety of the dataset versus subsets, such as the initial 10,000 entries per group. This raises a pivotal question about the balance between the quality and quantity of data—whether the focus should be on curating select, high-quality data or amassing a vast, average dataset. For instance, the distinction between gathering niche research papers versus the expansive content of Reddit presents a fascinating dichotomy to consider. The pursuit of this inquiry not only promises to refine our understanding of data's intrinsic value but also to enhance the efficacy of our computational models in handling classification challenges across diverse datasets.

Formulas

Conditional Probabilities

$$P_{\text{smooth}}(z|x,y) = \frac{\text{Count}(x,y,z)+\alpha}{\text{Count}(x,y)+\alpha\cdot N}$$

MLE

$$\hat{\theta} = \arg\max_{\theta} L(\theta;\text{data})$$

Bayesian Inference

$$P(H|D) = \frac{P(D|H)\cdot P(H)}{P(D)}$$

Distances

$$\text{Euclidean Distance: } d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

$$\text{Manhattan Distance: } d(x,y) = \sum_{i=1}^{n}|x_i-y_i|$$

$$\text{Minkowski Distance: } d(x,y) = \left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{\frac{1}{p}}$$