Language Modeling

Assignment Report

Dea Senka & Alba Murataj

Computer Science, University of New York in Tirana

Advanced Java, Fall 2022

Introduction

This report provides an explanation of the Language Modeling assignment for the Advanced Java course, which involves natural language processing (NLP). NLP is a field of artificial intelligence that deals with the interaction between computers and human languages, and aims to make it possible for machines to read, understand, and generate human-like language. In this assignment, we are required to use NLP techniques to analyze a mystery text file and identify its language by comparing it to a set of reference text files that are organized by language in subdirectories.

To complete the assignment, we must use the Java Streams API to process the text files concurrently and apply the cosine similarity method, which is a measure of the similarity between two non-zero vectors of an inner product space that determines the angle between them. The use of for loops should be avoided as per the project requirements. The report aims to give a clear and concise description of the steps involved in creating the language model using NLP techniques.

Implementation

The LanguageModel class in our Java program implements a basic natural language processing task of language classification using a language model in Java. The language model is constructed from a directory of text files, with each subdirectory representing a different language. The language model is represented as a histogram, which is a mapping of n-grams (sequences of n consecutive words) to their frequencies in the text.

The constructLanguageModel method is responsible for constructing the language model. It takes a directory as an input, and processes each language subdirectory and text file within it to create a histogram for the language. The histograms are then added to a mapping of languages to histograms, which represents the language model. The processTextFile method reads the text file, filters out punctuation, standardizes the words to lower case and tokenizes them into a list of words. It then processes the words in their n-gram sequence and updates the histogram by merging the n-gram and its frequency (1) using the merge method. If the n-gram is already in the histogram, its frequency is incremented by 1.

The constructLanguageModel method gets a list of all language subdirectories using File::isDirectory as a filter. It then creates a stream of the language subdirectories using Arrays.stream, and invokes the forEach method on the stream, passing a lambda as an argument.

The lambda is executed for each language subdirectory, and creates a histogram (a mapping of n-grams to their frequencies) for the language. It then gets a list of all text files in the language subdirectory using file -> file.getName().endsWith(".txt") as a filter, and creates a stream of the text files using Arrays.stream. It invokes the forEach method on the stream, passing a lambda as an argument.

The lambda is executed for each text file, and invokes the processTextFile method, passing the text file and the histogram as arguments.

To classify a text, the runClassification method is used. It takes a text file as input, and reads and processes the text in the same way as in constructLanguageModel. It then calculates the cosine similarity between the text histogram and each language model using the CalculateSimilarity method. The language with the highest similarity is selected as the classification.

The LanguageModel class makes use of streams and lambdas to process the text data and update the language models concurrently. The use of streams allows for efficient and concise processing of the data, while the use of lambdas enables the program to easily perform operations in parallel using multithreading.

Here are some examples of how streams and lambdas are used in the LanguageModel class:

Arrays.stream(languageFolders).forEach(languageFolder -> { ... }): This line uses a stream of languageFolders (an array of directories), and invokes the forEach method on the stream, passing a lambda as an argument. The lambda is executed for each element in the stream (in this case, for each language folder), and performs the operations inside the curly braces.

reader.lines().flatMap(line -> Stream.of(line.split("\\s+"))) ... .forEach(word -> { ... }): This chain of stream operations reads the lines of a text file, flattens the stream of lines into a stream of words, filters out punctuation and standardizes the words to lower case, and then invokes the forEach method on the resulting stream of words, passing a lambda as an argument. The lambda is executed for each word in the stream, and performs the operations inside the curly braces.

Arrays.stream(textFiles).forEach(textFile -> processTextFile(textFile, histogram)): This line uses a stream of textFiles and invokes the forEach method on the stream, passing a lambda as an argument. The lambda is executed for each

element in the stream and invokes the processTextFile method, passing the text file and the histogram as arguments.

After all text files have been processed, the constructLanguageModel method updates the language models with the histogram for the language.

Conclusion

The LanguageModel class in our Java program successfully implements a basic natural language processing task of language classification using a language model in Java. It constructs a language model from a directory of text files, and classifies a text by calculating the cosine similarity between the text histogram and each language model. The program makes use of streams and lambdas to efficiently and concisely process the text data and update the language models concurrently. This enables the program to perform language classification effectively and efficiently, making it a useful tool for a variety of natural language processing tasks.