NGram

1. Implementation

1.1. Architecture

The entire project contains 3 important modules.

1. Ngram
2. Distribution
3. NIO

1.1.1. Ngram

This module contains the basic Ngram helpers and algorithms. An Ngram model is defined as a tuple `(Either Int Int, Map a Int)`. `Either Int Int` decides the Ngram level, ie. either character level or word level. `Right` is mapped to character level and `Left` is mapped to word level. The polymorphic `Map a Int` is a basic frequency map, where in, for a character level Ngram `a` is `[Char]` and for a word level Ngram `a` is `[String]`.

Two important helper functions in this module are `chunksOf` and `freq`. `ChunksOf` is a polymorphic function which breaks a list into a list of list of n elements each and `freq` make a frequency map given a list of elements. `wordsP` is a modified words function that breaks a `String` into a list of words based on the given punctuation list. The most important function in this module is `predictN` which predicts the next `k` elements given a model.

1.1.2. Distribution

This module contains simple functions for discreet distributions. A distribution is represented simply as `[(a, Int)]` which is nothing but a frequency map. The function `sample` emulates the process of sampling from a random distribution.

1.1.3. NIO

This is the glue module which contain all the `IO` actions and hence is named `NIO` short for `Ngram IO`. `readCorpus`, `saveModel` and `readModel` are simple functions wrapped over basic file functions. This module also defines the `Command` data type which is are the valid application commands. `runCommand` runs the given command to produce some given effect.

1.2. Testing

The code contains basic unit tests to test the working of necessary functions. The functions that depend on randomness cannot be tested in the same style as unit tests. There is a `predictND` function in `Ngram` that outputs informative statements for `REPL` testing.

One error which was discovered during `REPL` testing was the random sampling. The emulated random sampling depends on random number generation. The random number should be generated between 1 and the size of the distribution. It was discovered that, initially, the random number generator produced numbers between 0 and size of distribution - 1.

1.3. Implementation choice in other languages

Personally, I think, python is the best language to use for any Machine learning algorithms because of its simplicity.

But for this project and NLP in general, I believe Haskell is the best choice for the following reasons.

1. It is easier and natural to model any kind of structure of a language in Haskell.
2. The structures and functions in Haskell makes it easier to make single pass functions with efficient folds.