Academic Poster : Ngram

Introduction

Language models, in a nutshell, are the type of models that assign probabilities to the sequences of words.

The simplest model that assigns these probabilities to sentences or sequence of words is N-Gram. An N-gram is a sequence of N words: a 2-gram (or bigram) is a two-word sequence of words like “please turn”, “turn your”, or ”your homework”, and a 3-gram (or trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”.

Code Snippets

```haskell
-- Break any list of elements into chunks of n
chunksOf :: Int -> [a] -> [[a]]
chunksOf i s = filter ((== i) . length) $ taker i s
  where
    taker _ [] = []
    taker i t@(x:xs) = take i t : taker i xs
```

```haskell
predictNext :: (Eq a, Ord a) => Model [a] -> [a] -> MaybeT IO [a]
predictNext (n, f) s = do
  let (tS', tS) = splitAt (length s - nSize n + 1) s
  fmap (tS' ++) . sample . filterD ((== tS) . init) . mkDistribution $ f

-- A function to predict the n characters/words depending on the model
predictN :: (Show a, Ord a) => Int -> Model [a] -> [a] -> IO [a]
predictN 0 _ s = return s
predictN k m s0 = do
  mS1 <- runMaybeT $ predictNext m s0
  case mS1 of
    Nothing -> return s0
    (Just s1) -> predictN (k - 1) m s1
```

```haskell
-- Choose i'th element of the distribution
choose :: Int -> Distribution a -> Maybe a
choose i [] = Nothing
choose i (x:xs)
  | i - snd x > 0 = choose (i - snd x) xs
  | otherwise = Just (fst x)

-- Size of the distribution
size :: Distribution a -> Int
size = foldr ((+) . snd) 0

-- Emulate the discrete random distribution and pick a random element
sample :: Distribution a -> MaybeT IO a
sample d = MaybeT $ flip choose d <$> randomRIO (1, size d)
```

Implementation

The entire project contains 3 important modules, namely, **Ngram**, **Distribution** and **NIO**.

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.

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.

NIO

This is the glue module which contain all the IO actions, Eg. readCorpus, saveModel and readModel. 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.

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.

Implementation Choice

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