Predicting next word

Guillaume date: 2015-08-23

Introduction

For this capstone project, we were asked to create a shiny app for predicting the next word.

- Presentation of the algorithms
- Application presentation
- Futur of the app

N-grams Algorithms

- The simple, unsmooth, n-grams.
  The general equation for the N-gram approximation to the conditional probability of the next word is:

$$P\big(\mathcal{W}_n \mid \mathcal{W}_1^{n-1}\big) \approx P\big(\mathcal{W}_n \mid \mathcal{W}_{n-N+1}^{n-1}\big)$$

  This method while simple and effective, is limited when facing unknown but valid sequence or low probabilities. Smoothing alievate this limitation.

- Laplace smoothing (add one count), Good-Turing smoothing to name 2 of the most known.

- Interpolation and backoff are two additional technics to enhance the algorithms.

Algorithm used - Kneser-Ney Algorithm

The best explanation of the algo is from the book Speech and Language Processing.
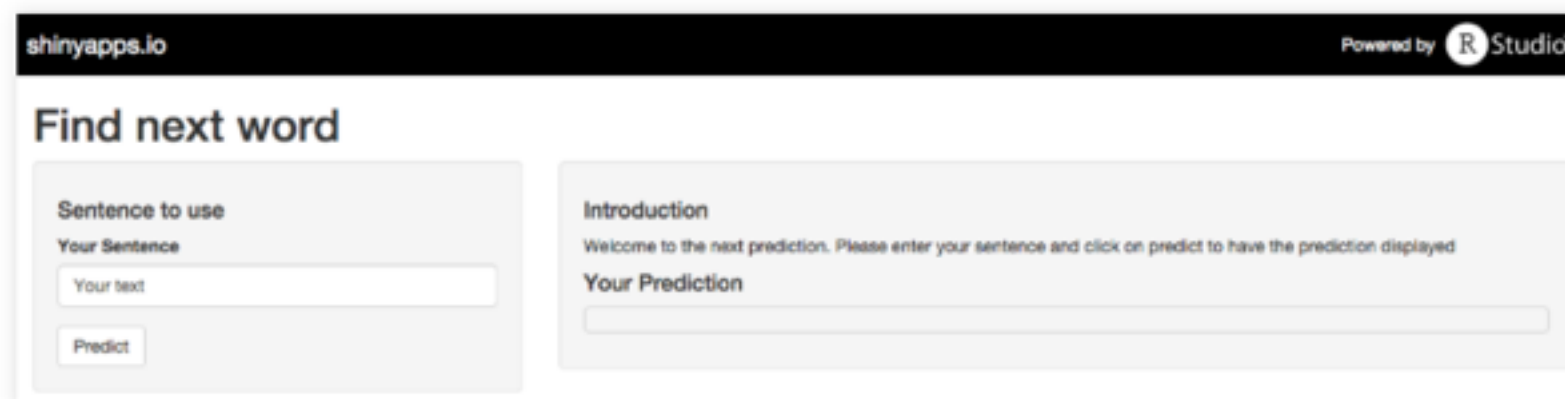The KN intuition is to base our estimate on the number of different contexts word w has appeared in. Words that have appeared in more contexts are more likely to appear in some new context as well. We can express this new backoff probability, the "continuation probability", as follows

$$
P_{KN}(w_i \mid w_{i-1}) = \begin{cases} \dfrac{C(w_{i-1}w_i)-D}{C(w_{i-1})}, & \text{if } C(w_{i-1}w_i) > 0 \\[2em] \alpha(w_i)\, \dfrac{\left|\{w_{i-1}:C(w_{i-1}w_i)>0\}\right|}{\sum_{w_i}\left|\{w_{i-1}:C(w_{i-1}w_i)>0\}\right|} & \text{otherwise.} \end{cases}
$$

How to use the app

- The application is pretty straighfoward

- enter a sentence and click predict button to predict the next word

- It's using a 3-grams Kneser-Ney algorithm

- Migrate the algorith to a programming language that can be compiled to use in an app

- Tune the algorithm with larger corpus