Word-O-Matic 3000

JNabonne

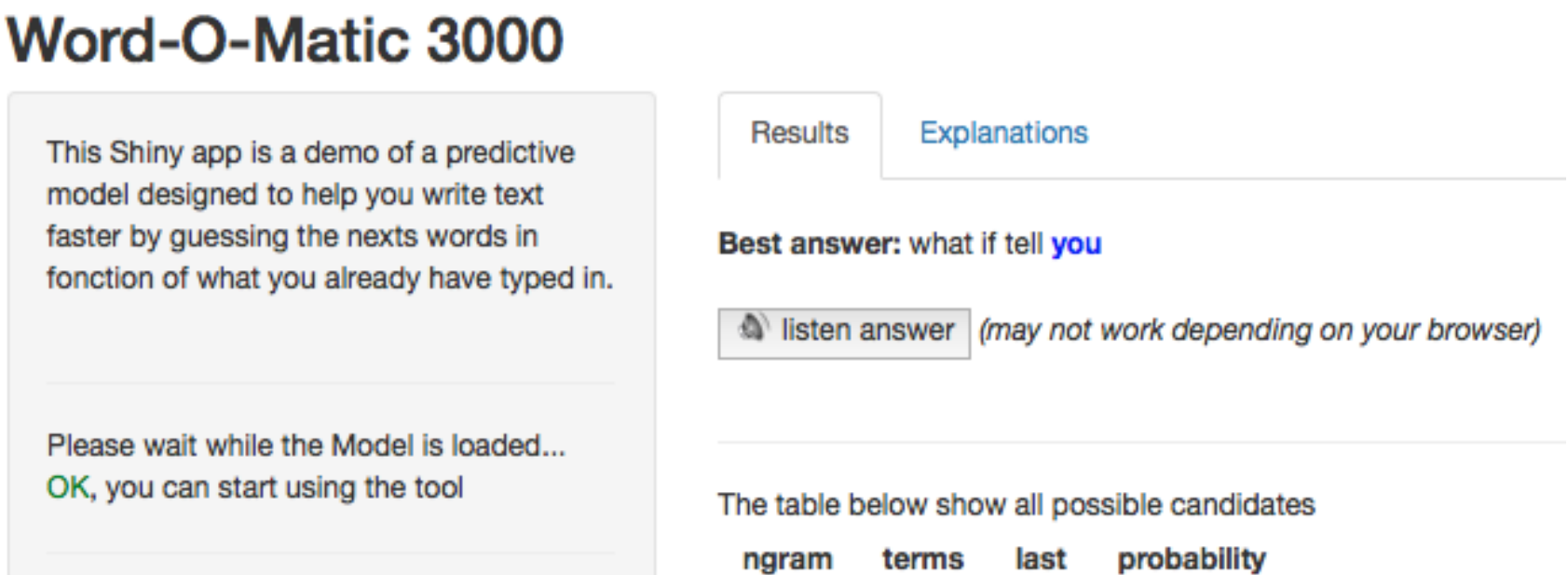
04/02/2019

Context / About

- Capstone project for **JHU Data Science Specialization**
- **Goal:** to create a predictive model to guess next word
- **Dataset given by SwiftKey** with english text from:
  1. Blogs (~210MB)
  2. News (~205MB)
  3. Twitter feed (~170MB)
- Very little guidance from professors *(just the main objective, couples of links about NLP, ngram and backoff algorithm and some warnings about memory challenges!)*
- **Very challenging and rewarding: learn a lot!**

Result: a Shiny app to test the predictive model

- The application is available on shiny server here
- quite straight forward:
  1. start typing a sentence in the left section
  2. wait a few second to get some proposition (on the right side)
- app will give you:
  - the best answer from my algo (cf. next slide for more detail)
  - as a bonus, a text-to-speech button to hear it
  - if any, other alternative answers found by the model
- there is also a special tab briefly explaining the app



The Model and the algorithm behind

Following the EDA (cf. milestone report), the `quanteda` library has been used to create various ngram (from 1-grams up to 5-grams) which are manually enriched with statistics and probability calculus ; this is the base of the model.

The algorithm used is a stupid-backoff version *(a simplified version of the Katz'one)* that, when given a sentence, will:

1. count the number of words `n` in it
2. look in the (n+1)-gram data if it can find the next word
3. if not it will 'back-off': take the last (n-1) words of the input and look in the n-gram data
4. if not, back-off again and again until either finding an answer or ending up at unigram level (1-gram)
5. the algo will return the best candidate possible

Annex: GUI and one Example

Example if you type in 5 words "please father I want to" > *it will only keep "father I want to"'</i>" and look in 5-grams* > if it does not find a candidate word, it will look in 4-grams with "I want to" > *then if necessary into 3-grams with "want to"* > then if necessary into 2-grams with "want to" *> if nothing works, it will dumbly return the most probable word from the corpus