Capstone: Improving NLTK Lemmatization

Presented by Shannon Bingham

Powered by 

Executive Summary

The Natural Language Toolkit (NLTK) is a widely used open source solution for language processing.

In my previous work with its interface to the WordNet lexical reference library, I became deeply curious about how it worked.

My questions led me to improve the product in my test environment.

These questions drove my thinking:

- If x% of words in written English make up a large y% of corpora, can we leverage that understanding prior to feature selection and modeling?

- How can I improve my NLP input data?

- New words are added to English fairly often these days. How can we keep our NLP models current?

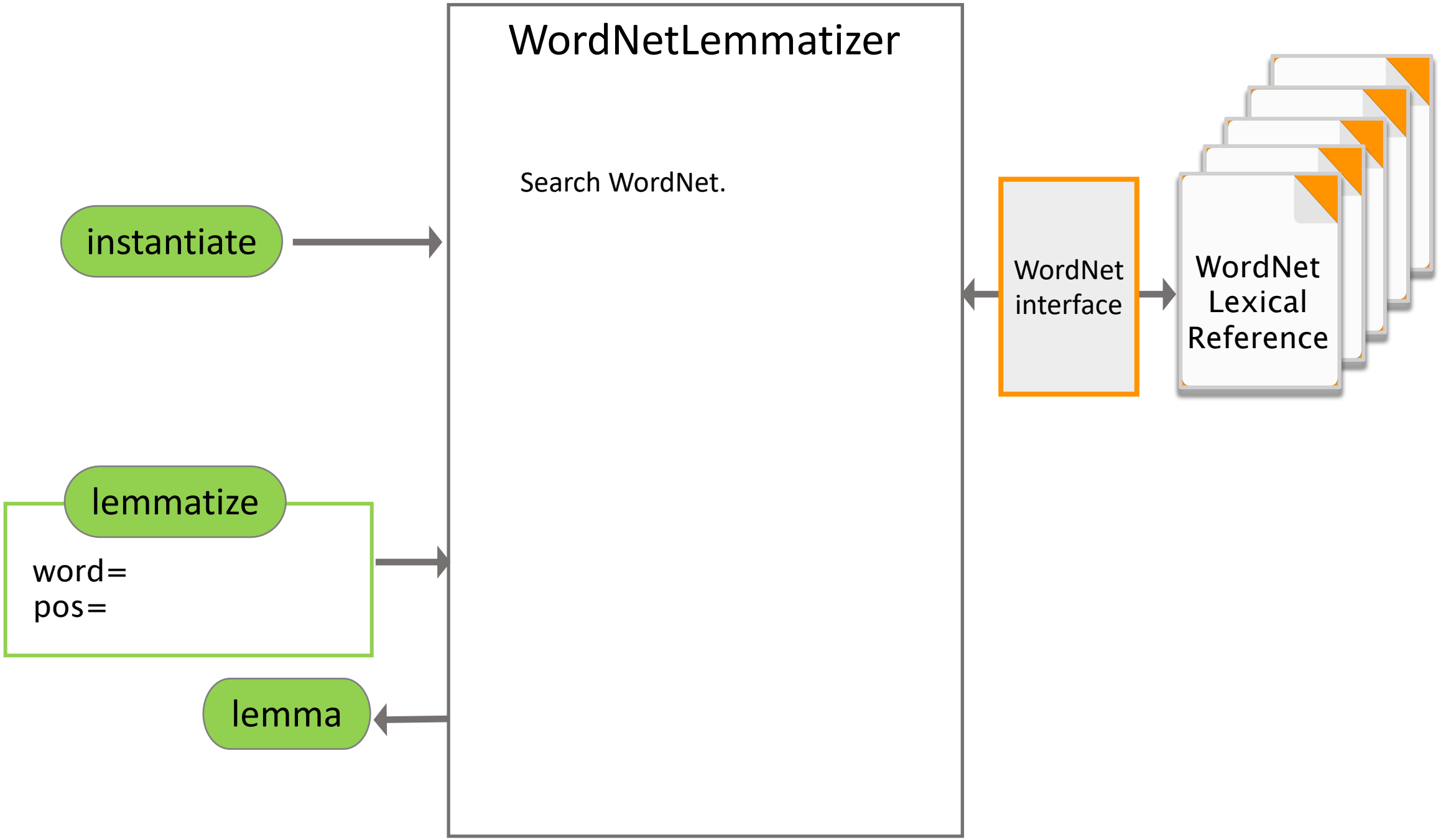
My work followed the data science process.

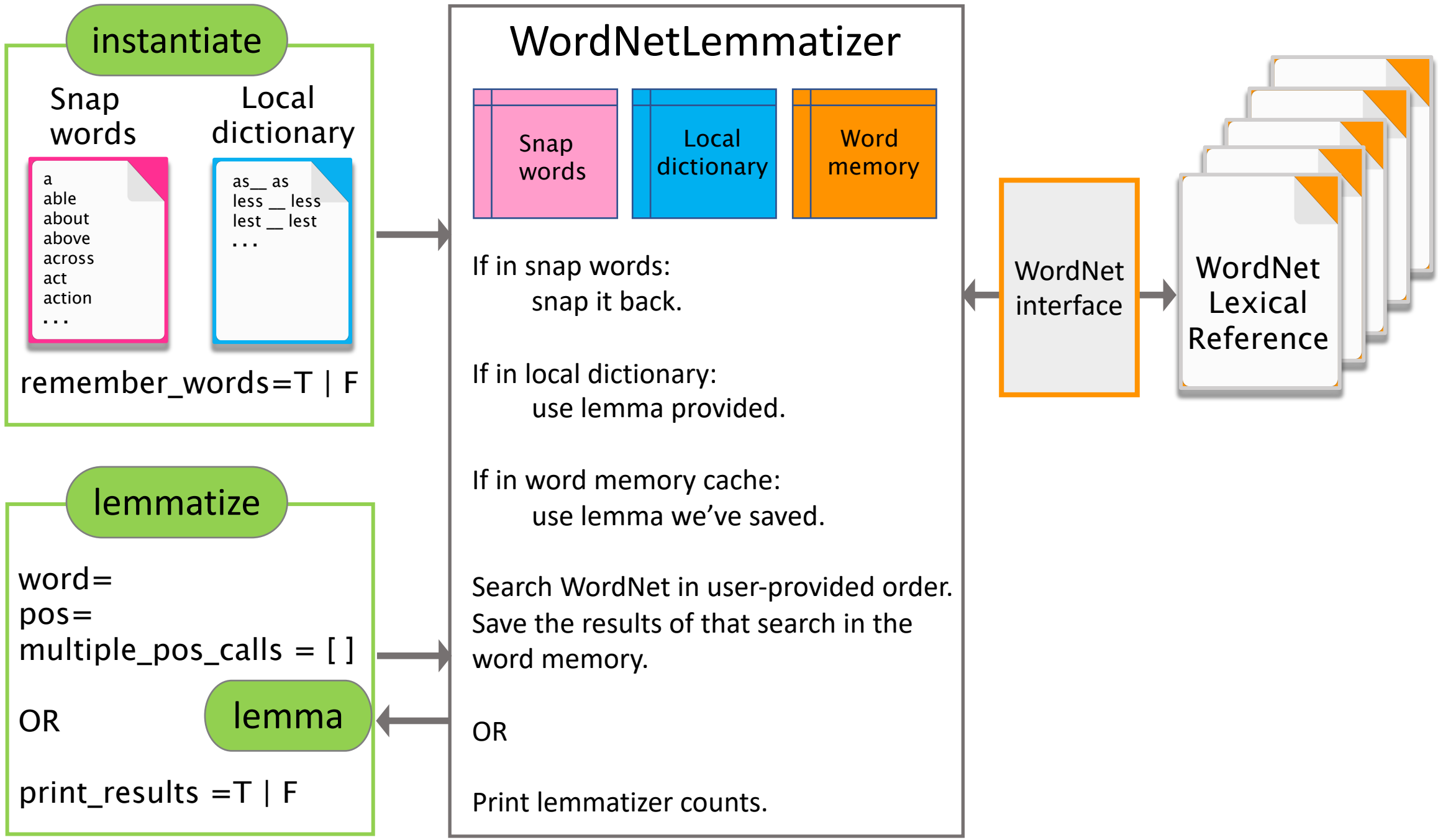
- Problem definition:

  Focus on functionality and performance.

- Data collection:

  IMDB reviews (http://ai.stanford.edu/~amaas/data/sentiment/)


WordNetLemmatizer
Search WordNet.
instantiate
lemmatize
word=
pos=
lemma
WordNet interface
WordNet Lexical Reference


instantiate
Snap words
Local dictionary
a
able
about
above
across
act
action
...
as__ as
less __ less
lest __ lest
...
remember_words=T | F
lemmatize
word=
pos=
multiple_pos_calls = [ ]
OR
lemma
print_results =T | F
WordNetLemmatizer
Snap words
Local dictionary
Word memory
If in snap words:
snap it back.
If in local dictionary:
use lemma provided.
If in word memory cache:
use lemma we've saved.
Search WordNet in user-provided order.
Save the results of that search in the word memory.
OR
Print lemmatizer counts.
WordNet interface
WordNet Lexical Reference

The hit rate on the high frequency words stayed constant. The hit rate on the saved words made a big difference.

```
1  # Get results from the Lemmatizer.
2  lemmatizer.lemmatize(' ', print_results=True)
```

```
-------------------------------------------------
Type                      Number      Hits % Total
-------------------------------------------------
snapwords                    807     13797   56.40
local dictionary              16       225    0.90
remembered words            4264      6165   25.20
* not cached *                 0      4264   17.40

Total                                24451
```

```
1  # Get results from the Lemmatizer.
2  lemmatizer.lemmatize(' ', print_results=True)
```

```
-------------------------------------------------
Type                      Number      Hits % Total
-------------------------------------------------
snapwords                    807    133396   56.90
local dictionary              16      2141    0.90
remembered words           16994     82078   35.00
* not cached *                 0     16994    7.20

Total                               234609
```

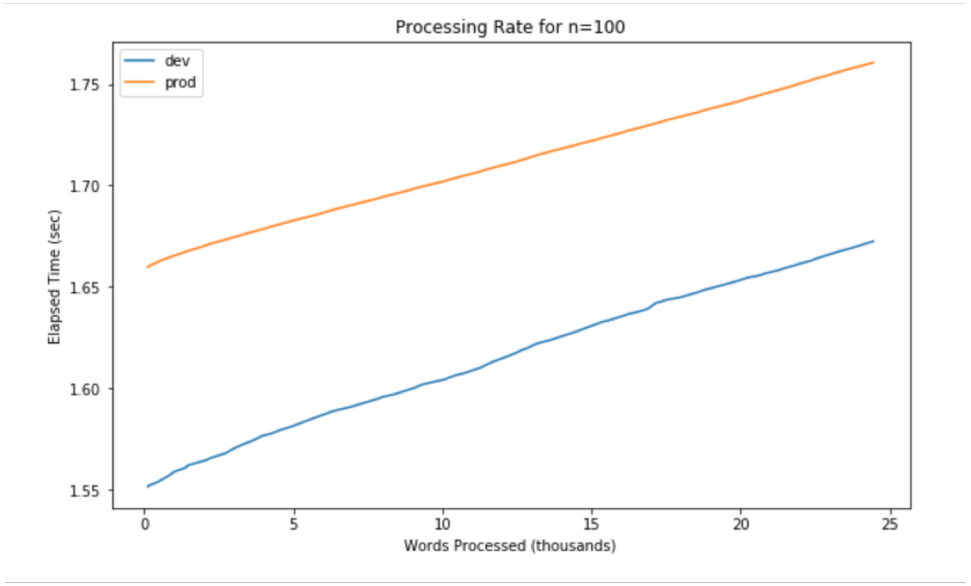
```
1  # Get results from the Lemmatizer.
2  lemmatizer.lemmatize(' ', print_results=True)
```

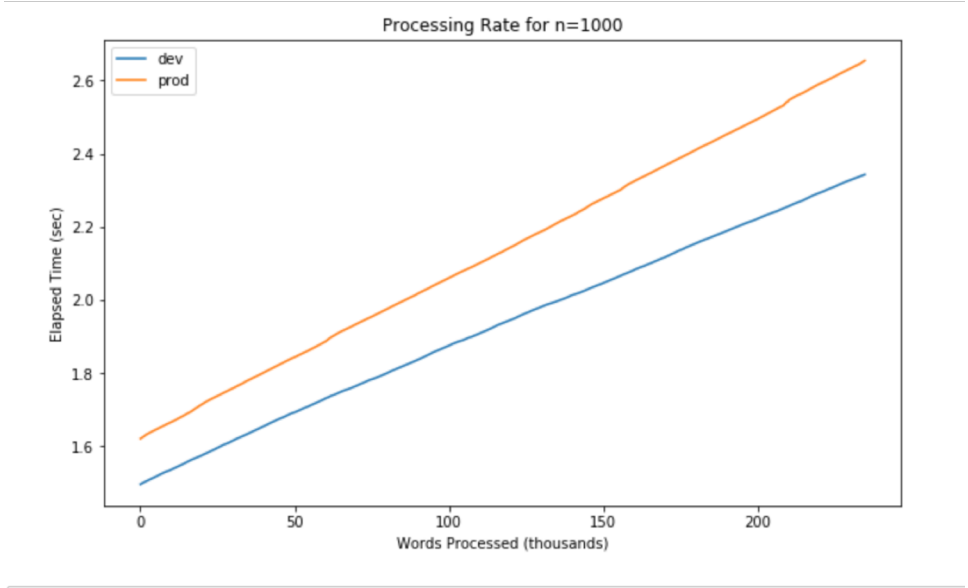
```
-------------------------------------------------
Type                      Number      Hits % Total
-------------------------------------------------
snapwords                    807   1342413   57.10
local dictionary              16     20799    0.90
remembered words           50131    936467   39.90
* not cached *                 0     50131    2.10

Total                              2349810
```

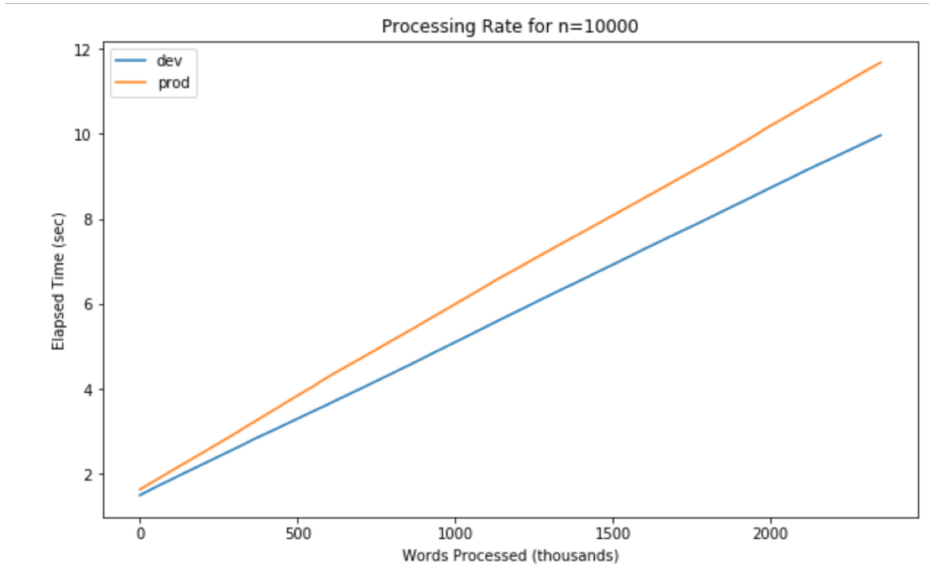
The model using the dev module tracked with the production module.

	SGDClassifier	MultinomialNB	SIA w/GaussianNB
N=100	Prod accuracy .5 Dev accuracy .63	Prod accuracy .73 Dev accuracy .76	Prod accuracy .7 Dev accuracy .7
N=1000	Prod accuracy .8 Dev accuracy .83	Prod accuracy .71 Dev accuracy .73	Prod accuracy .7 Dev accuracy .71
N=10,000	Prod accuracy .87 Dev accuracy .87	Prod accuracy .85 Dev accuracy .84	Prod accuracy .7 Dev accuracy .70

The dev version ran a bit quicker than the production version in my testing.







My conclusion is that:

The functionality increased with no loss of performance or predictive power.

Next steps:

Indepth tuning.

Presented by Shannon Bingham