LANGUAGE DETECTION

COREY MANDERS APRIL 2020

STRATEGY USED

- Use frequency tables to detect the language
    1. Split text into words
    2. For each word, check if it is in the top N words in the language
    3. Add the number of “hits” for the words in the list
    4. Pick the language which results in the highest number

COMPETING STRATEGIES

IS THIS THE ONLY WAY TO DO IT?

- Definitely not. For example the paper suggested using character unigrams, bigrams, or trigrams, and then computing the KL distance of a testing sequence with the histogram/ probability function generated with a training set.
- Another approach may be to use something like an RNN to do the categorisation, probably at a character level
- There are several “older” Machine Learning approaches. For example create a vector of meta data (character unigrams, bigrams, word lengths, etc.), and classify that vector using SVM, random forest, etc.,
- There are likely ways of combining these approaches, and many others

COMPETING STRATEGIES

WHY DID I CHOOSE THIS APPROACH

- When exploring the problem, I came across https://github.com/hermitdave/FrequencyWords, realised it had all of the frequency counts for the languages from the data set, and saw it as easy to implement.
- Had confidence that frequency counts could lead to accurate predictions
- Realised that the approach was simple, and would likely work for many situations

COMPETING STRATEGIES

STRATEGY DOWNSIDES

- The strategy needs at least a few “words” from the language to make an accurate prediction.
- Using a completely character-based approach (like character bigrams, trigrams, etc.), for many languages, may produce accurate results with less input data needed.
- Using a character-based approach may be less computationally complex.
  - Computing histograms is computationally fast
  - Computing KL distance, or projection of a vector into a multi-dimensional space is O(1), etc. for SVM, could be computationally fast.

CONTINUING DEVELOPMENT

WHAT WOULD I TRY NEXT

- I would likely try a character based approach (unigrams, then bigrams, followed by KL distance)
- I would try creating a meta-data vector, and using SVM, or random forest , etc. Simply because if you set up the data, training, testing, for one method, the consistency of the scikit-learn interface allows testing of several ML techniques easily.
- I would try an RNN or CNN for detection, it would probably work well, especially if there was additional data.
- Would imagine, the winning strategy may have a combination of techniques

ACCURACY

HOW DID THE STRATEGY PERFORM?

Text sequences: 5, 15, and 15 word sequence

Overall average on all languages:

- **71.7% (5 words given)**
- **93.7% (15 words given)**
- **97.6% (30 words given)**

ACCURACY

HOW DID THE STRATEGY PERFORM?

Text sequences: 5, 15, and 15 word sequence (note there is variability given random choice of word sequences)

Strong performance on Cyrillic character sets (EL, BG) :

- Easily get 100% with 30 words

Weaker performance on English, which can be mistaken for languages such as NL, because of the character set being the same.

ACCURACY

HOW DID THE STRATEGY PERFORM?

Text sequences: 5, 15, and 15 word sequence (note there is variability given random choice of word sequences)

Strong performance on EL :

- 96% (5 words given)
- 100% (15 words given)
- 100% (30 words given)

ACCURACY

HOW DID THE STRATEGY PERFORM?

Sentences: minimum 5 words, average sentence length 20.4 words

Overall average on all languages:

- **97.14%**

- **9 languages had 100% accuracy, 50 trials for each language (random sentences chosen)**
- **Lowest accuracy 86%, RO, majority of errors are from sentences with low numbers of words.**