Language Identification for (very) short texts

•••

Alcméon

It’s a solved problem

“Written language identification is regarded as a fairly easy problem”

“N-gram-based text categorization”, *Cavnar, William B., and John M. Trenkle*, 1994

“Statistical identification of language”, *Dunning, Ted*, 1994.

“Language identifier: A computer program for automatic natural-language identification of on-line text”, *Beesley, Kenneth R*, 1998.

Practical Open Source Solutions

https://github.com/optimaize/language-detector

- 71 languages covered
- Apache licence
- Java-based

https://github.com/peterc/whatlanguage

- 19 languages covered
- MIT licence
- Ruby-based

Many others...

But, ...

“This software does not work as well when the input text to analyze is short, or unclean. For example tweets.”

“It works [...] very poorly on short or Twitter-esque text”

How do these things work ?

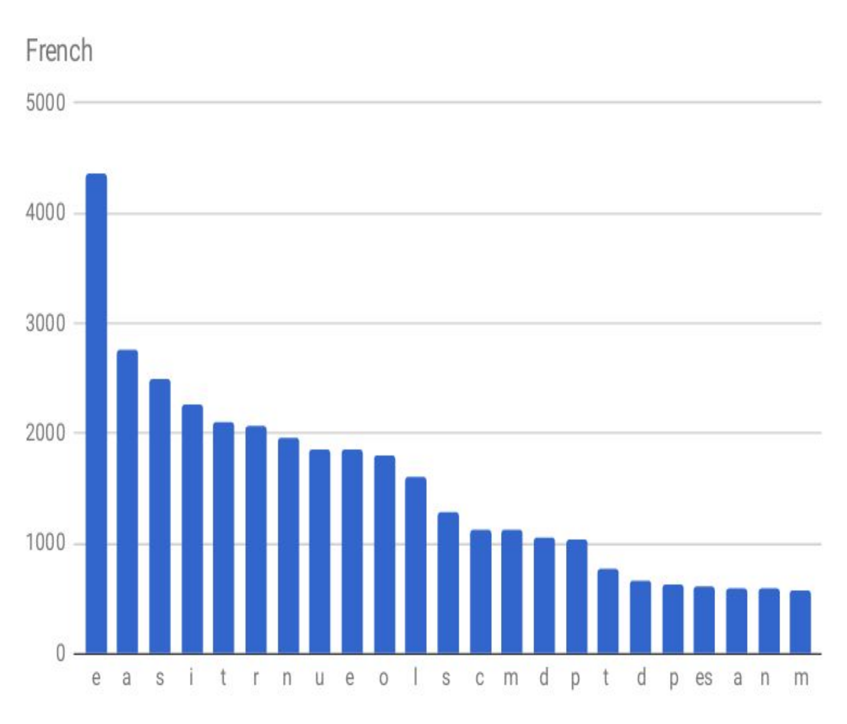
1. For language reference corpus, generate n-gram distribution, keep k most-frequent n-grams

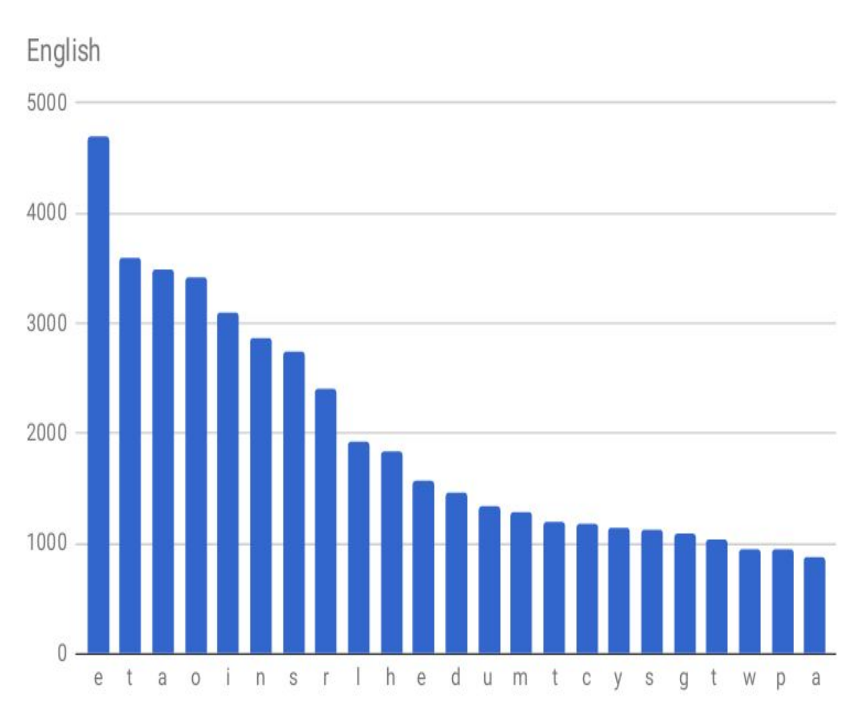
2. For new unknown text, generate n-gram distribution

3. Calculate out-of-place distance to each reference language distribution

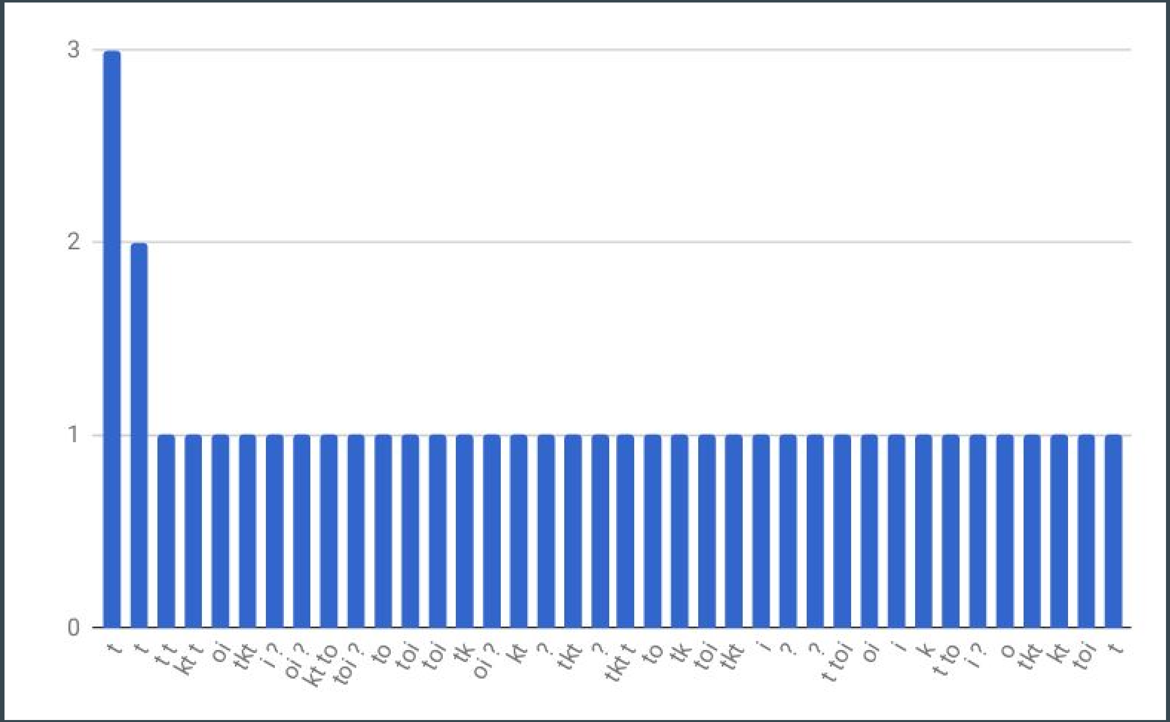
4. Choose “closest” reference language distribution

Reference n-gram distributions





“Tkt toi ?” ngrams



“Tkt toi ?” out-of-place distance

English	87300
Dutch	87541
French	87553
Italian	87753
German	87926
Spanish	88444
Russian	88845
Arabic	90000

Summary

N-gram distributions are very coarse on short text

SMS-speak n-gram distributions are very strange

But it’s a classification problem !

Input features: ngram counts

Output class: language

Training+test data: twitter sample stream :)

Algorithms: Naive Bayes + kbest/chi2

Implementation: scikit-learn

Download the code: https://github.com/mathieu-lacage/sophiaconf2017

Prepare the data

1. Download the data:
    a. Create a twitter app (https://app.twitter.com)
    b. Generate access tokens
    c. Run twitter-data.py for a while
2. Preprocess tweets
    a. Run preprocess.py
3. Extract features:
    a. Run extract-features.py
4. Output:
    a. X.mtx, y.npy, y-textcat.npy
    b. classes.json, features.json

Generate a dumb model

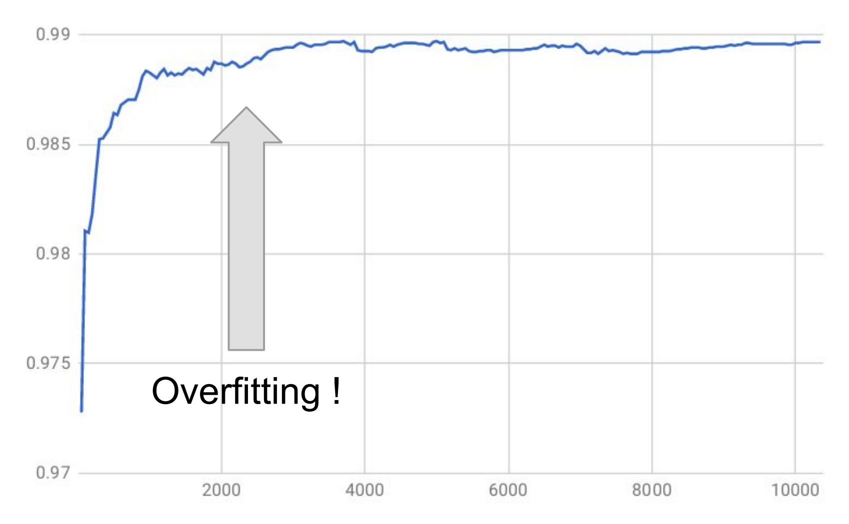
```
X = io.mmread('X.mtx')

y = numpy.load('y.npy')

select = SelectKBest(chi2, k = 1000)

classifier = MultinomialNB()

classifier = Pipeline([('kbest', kbest), ('nb', nb)])

classifier.fit(X, y)

joblib.dump(classifier, 'model.pkl')
```

Predict “tkt toi ?”

```
vector = [0] * len(features_by_name)
ngrams = Ngrams.generate(content)
for ngram, count in ngrams:
    if ngram in features_by_name:
        vector[self._features_by_name[ngram]] = count
x = numpy.array([vector])
pred = model.predict(x)
print classes_by_id[pred[0]]
```

Search for the right k

```
for k in range(50, 15000, 50):
    classifier = Utils.kbest_naive_bayes(k)
    scores = cross_val_score(classifier, X, y)
    print k, scores.mean()
```



What is missing for a production solution

1. More data
2. Better data (review twitter ground truth)
3. More languages
4. Check language class imbalance
5. Boost short messages
6. Test other classifiers, other feature selection criteria
7. Exploit more features (author profile lang)
8. Embed in a microservice

Summary

TextCat: 4% error rate

Dumbest possible classifier: 1% error rate

You will do much better with real data

We are hiring :)

Questions ?