Language detection in 7 minutes

Paweł Kubiak

pawel.kubiak@protonmail.com

PyConPl'20

Language Detection. Why?

Before we start any analysis of the text we need to know its language. We need to know it to decide which dictionary to download, which language model to load or which algorithm to use. Language is **must-have**.

Corpus collecting

In order to train our model, we will need a lot of text in different languages. For example, we can use:

- European Parliament Proceedings Parallel Corpus (21 langs)
- Universal Declaration of Human Rights (524 langs)
- Books / documents from WikiBooks project (76 langs)
- Wikipedia articles (314 langs)

Collect Wikipedia articles

We use MediaWiki API [1] to extract article plaintext:

```python
import requests

def get_wikipedia(lang: str, title: str) -> str:
    """Extract wikipedia article as plaintext using API."""
    req = requests.get(
        f"https://{lang}.wikipedia.org/w/api.php",
        params=dict(
            action="query", format="json", prop="extracts",
            titles=title, explaintext=True, redirects=True
    )).json()

    return list(req['query']['pages'].values())[0]['extract']
```

Sample Wikipedia article

```python
kapibara = get_wikipedia('pl', 'kapibara')
print(kapibara[:400])
```

```
Kapibara (Hydrochoerus) – rodzaj ssaka z podrodziny kapibar (Hydrochoerina
e) w rodzinie kawiowatych (Caviidae).


== Zasięg występowania ==
Rodzaj obejmuje gatunki występujące w Ameryce Środkowej (Panama) i Południo
wej (Kolumbia, Wenezuela, Gujana, Brazylia, Ekwador, Peru, Boliwia, Paragwa
j, Argentyna i Urugwaj).


== Morfologia ==
Długość ciała (bez ogona) 1025–1340 mm, długość ogona 10–20 mm, dł
```

Text cleaning

```python
import re

def clean(text):
    text = re.sub('[_\W\d]', ' ', text)  # non-alphanumeric and digits
    text = re.sub('\s+', ' ', text.strip())  # normalize whitespaces
    text = text.lower()  # convert to lowercase

    return text

czysta_kapibara = clean(kapibara[:300])
print(czysta_kapibara)
```

```
kapibara hydrochoerus rodzaj ssaka z podrodziny kapibar hydrochoerinae w ro
dzinie kawiowatych caviidae zasięg występowania rodzaj obejmuje gatunki wys
tępujące w ameryce środkowej panama i południowej kolumbia wenezuela gujana
brazylia ekwador peru boliwia paragwaj argenty
```

What are N-grams?

> *An N-gram is a contiguous sequence of n items from a given sample of the text. The items can be phonemes, syllables, letters or words according to the application.*

— Wikipedia, The Free Encyclopedia [2]

What are N-grams?

- (*unigram*) → p, y, c, o, n
- (*bigram*) → py, yc, co, on
- (*trigram*) → pyc, yco, con

N-grams in Python

```python
def make_ngrams(text: str, n: int) -> List[str]:
    """Generate all n-grams of length `n` from `text`."""
    for i in range(n, len(text)+1):
        yield text[i-n:i]

list(make_ngrams("pycon2020", 3))
```

```
['pyc', 'yco', 'con', 'on2', 'n20', '202', '020']
```

Top N-grams

> *The top 300 or so N-grams are almost always highly correlated to the language. ... Starting around rank 300 or so, an N-gram frequency profile begins to show N-grams that are more specific to the subject of the document.*

— N-Gram-Based Text Categorization [3]

N-grams ranking in Python

```python
from collections import Counter

def ranking(text: str, topn: int) -> List[str]:
    """
    Collect N-grams of given lengths from `text`.
    Return `topn` most common N-grams.
    """
    counts = Counter()
    words = text.split()
    for length in [1, 2, 3]:
        for word in words:
            counts.update(make_ngrams(f"_{word}_", length))

    return [key for key, _ in counts.most_common(topn)]
```

Sample N-grams rankings for PL

```python
text = clean(get_wikipedia('pl', 'Linux'))

rank = ranking(text, 200)
print(', '.join(rank))
```

```
_, a, e, i, o, n, r, s, t, w, z, y, c, u, m, k, p, d, l, j, e_, a_, ie, ni,
st, g, _p, in, _s, na, i_, ow, _w, li, er, _n, b, h, y_, te, ch, _z, o_, ż,
ro, ra, _l, ó, ą, po, an, h_, _o, _i, _d, wa, ch_, ko, _t, nie, w_, _na, m
_, nu, _li, ie_, ł, f, ny, ac, _m, pr, _k, lin, inu, ę, wi, za, cz, je, ne,
yc, rz, ze, _j, _po, ys, em, x, z_, do, _g, u_, yst, ia, ych, mi, na_, _pr,
ów, al, ej, t_, zy, or, od, ą_, ś, _u, _a, on, owa, op, to, ta, ic, en, x_,
ar, os, cj, _i_, ux, _r, sz, ki, wan, _je, nux, wy, le, ty, om, _ko, _w_, y
m, es, gr, aj, mo, _za, sy, ją, ani, go, ste, eg, j_, tem, wn, ak, si, ów_,
ter, ć, ks, dz, ć_, _z_, la, ux_, _sy, sys, rze, pro, ne_, tr, so, we, ię,
ka, rac, _do, em_, ob, ed, wo, ró, ę_, uż, est, go_, ej_, dy, sa, ma, ego,
sta, fi, me, st_, am, ec, ry, is, nyc, ym_, sk, _op, era
```

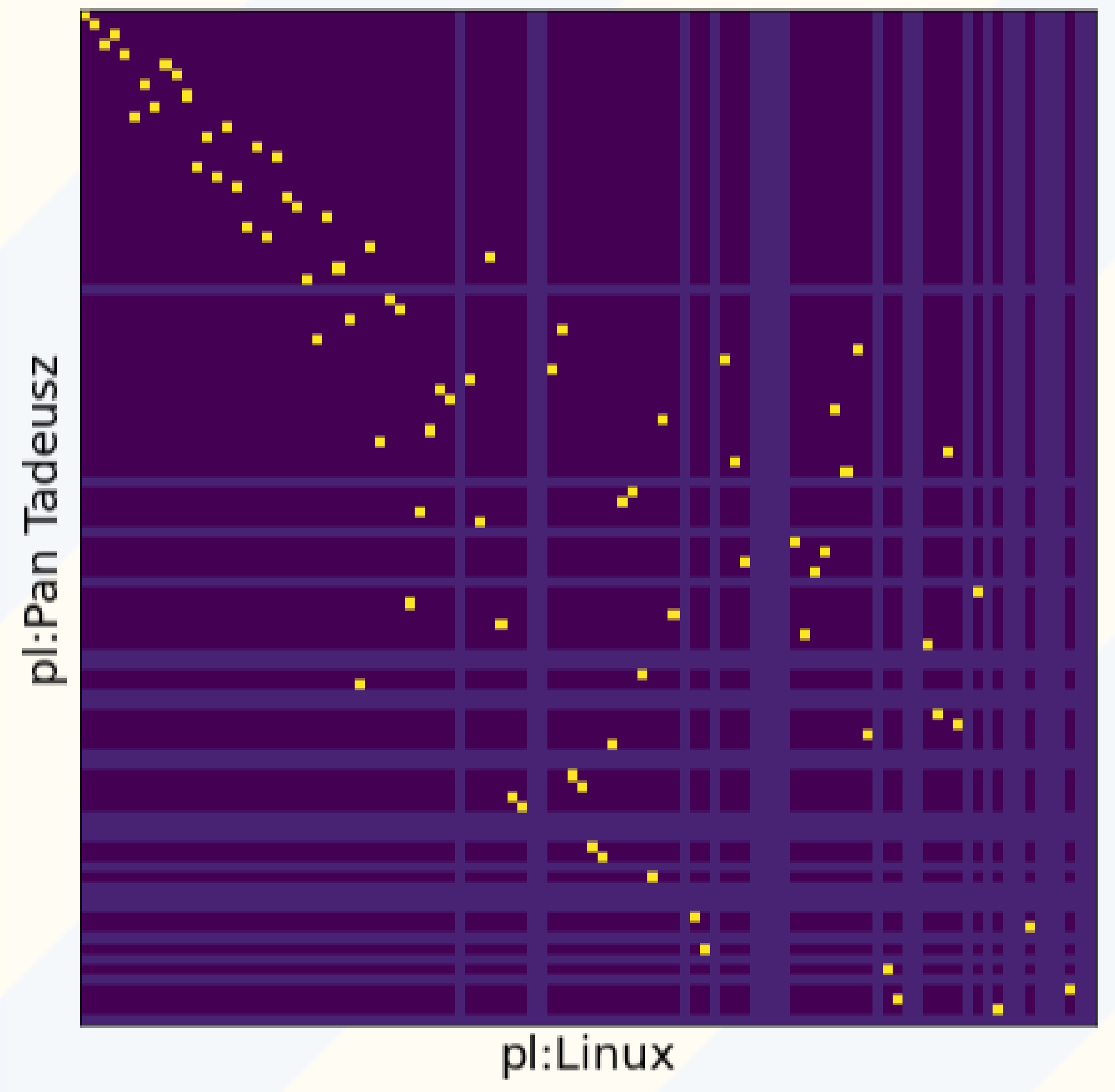
PyConPl'20

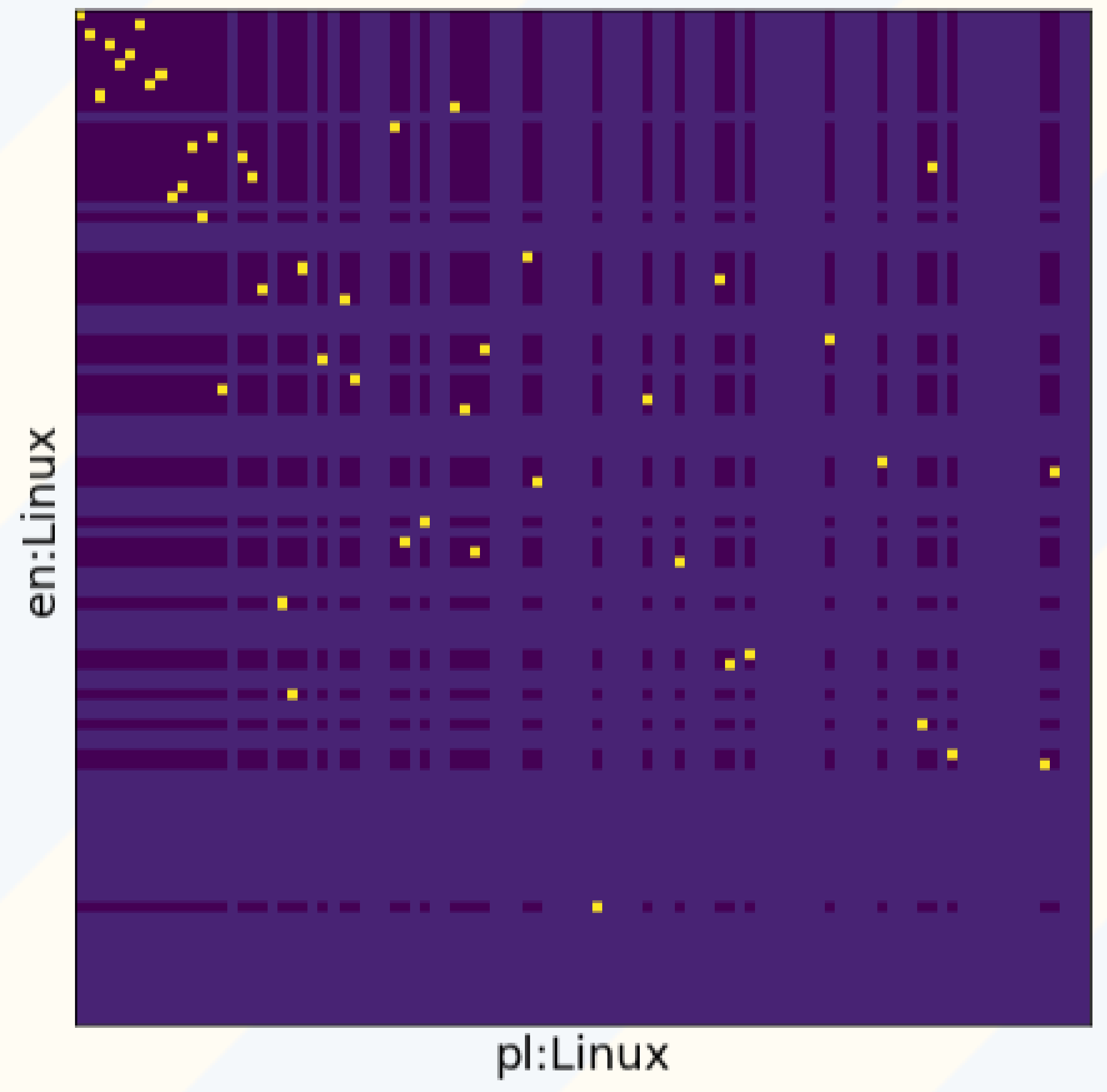
Hypothesis

> *Texts in the same language generate similar N-gram rankings, while for different languages the rankings will differ strongly.*

Rankings correlation

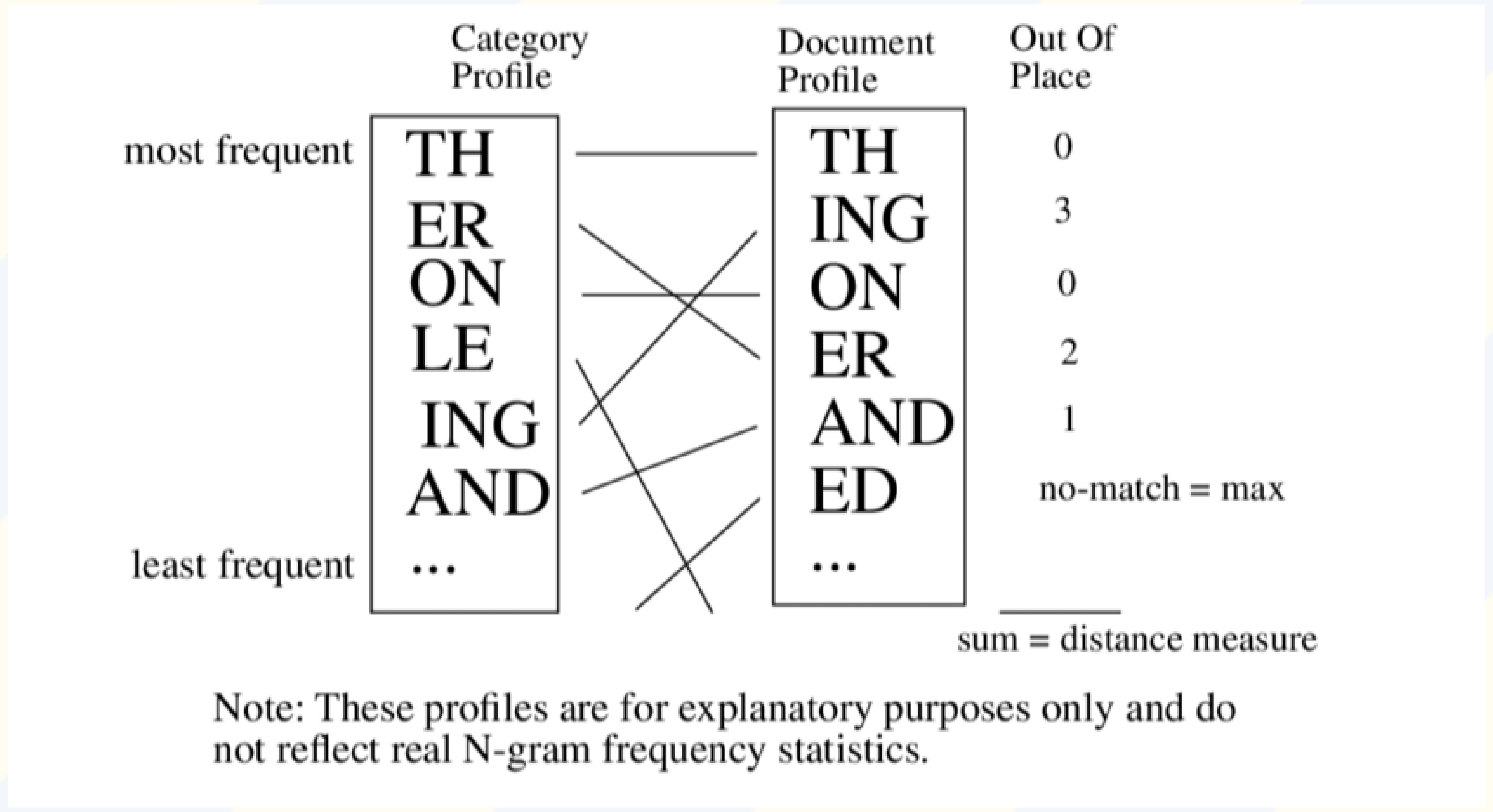
```python
_, ax =plt.subplots(1, 2, figsize=(16,8))
vizualize_rankings(ax[0], 'pl:Linux', 'pl:Pan Tadeusz')
vizualize_rankings(ax[1], 'pl:Linux', 'en:Linux')
```





PyConPl'20

Rankings Distance Measure



Note: These profiles are for explanatory purposes only and do not reflect real N-gram frequency statistics.

— N-Gram-Based Text Categorization [3]

Rankings Distance Measure in Python

```python
def distance(rank1: List[str], rank2: List[str]) -> int:
    """Compute distane between two N-grams rankings."""
    score = 0
    for i, key in enumerate(rank1):
        if key in rank2:
            score += abs(rank2.index(key) - i)
        else:
            score += len(rank2) # MAX penalty
    return score
```

```python
(
    distance(rank_wiki('pl:Linux'), rank_wiki('pl:Pan Tadeusz')),
    distance(rank_wiki('pl:Linux'), rank_wiki('en:Linux'))
)
```

```
(14956, 23623)
```

PyConPl'20

N-grams Language Detector

```python
class LinuxLanguageDetector:
    """Detect text language based on N-grams model."""
    def __init__(self, langs: List[str], n: int = 200):
        self.n = n
        self.ranks = {
            lang: rank_wiki(f"{lang}:Linux", n)
            for lang in langs
        }

    def detect(self, text: str) -> str:
        rank = ranking(clean(text), self.n)
        return min(
            self.ranks,
            key=lambda lang: distance(rank, self.ranks[lang])
        )
```

Testing

```python
langdet = LinuxLanguageDetector(['pl', 'en', 'de', 'fr', 'it'])
```

```python
langdet.detect("PyCon PL 2020 to trzynasta edycja "
    "ogólnopolskiej konferencji z grupy PyCon.")
```

```
'pl'
```

```python
langdet.detect("Python è un linguaggio di programmazione di più "
    "alto livello rispetto alla maggior parte degli altri linguaggi.")
```

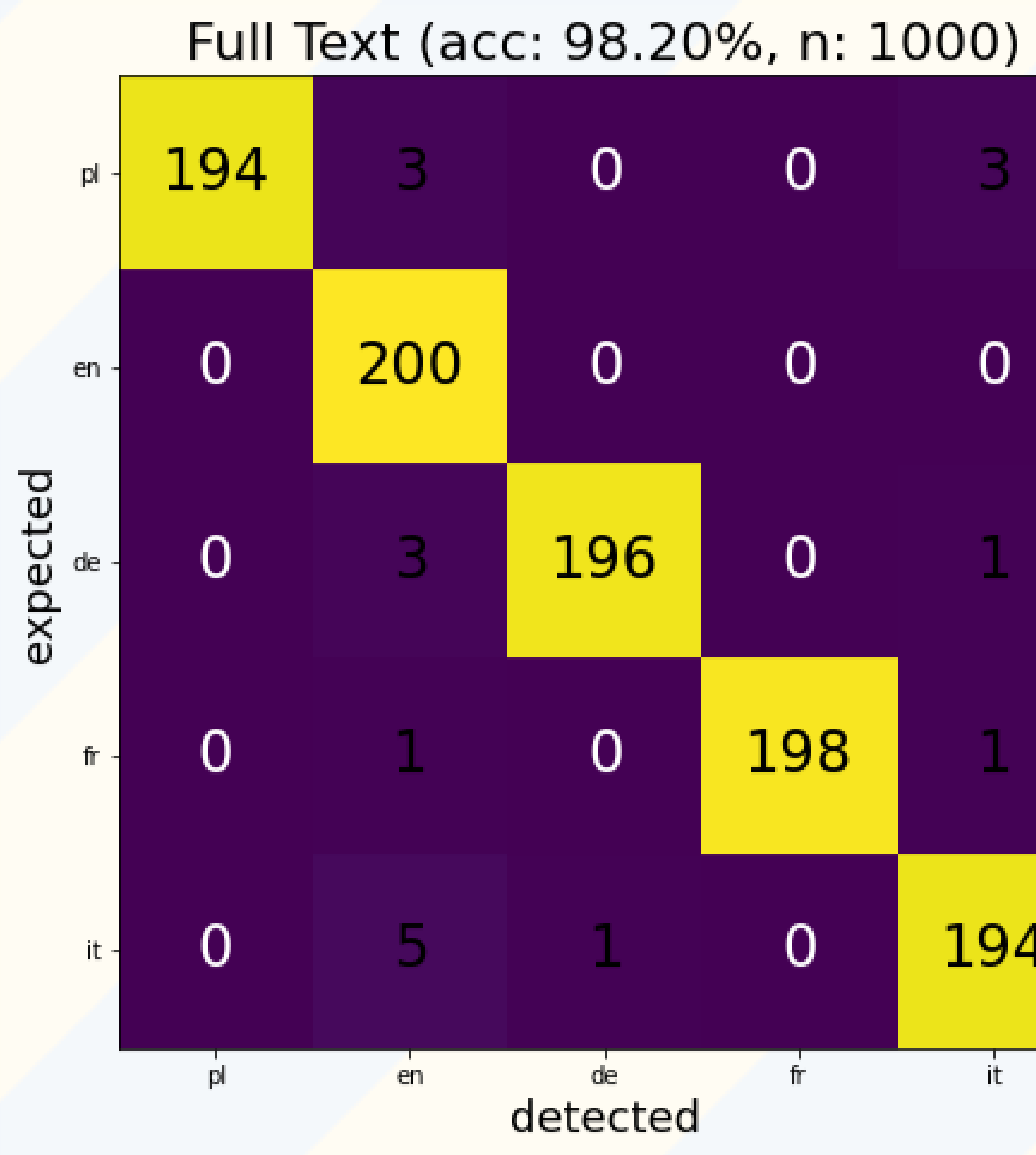
```
'it'
```

```python
langdet.detect("Capybaras are highly social, living in groups of up "
    "to 100 and communicating through a variety of vocalizations.")
```

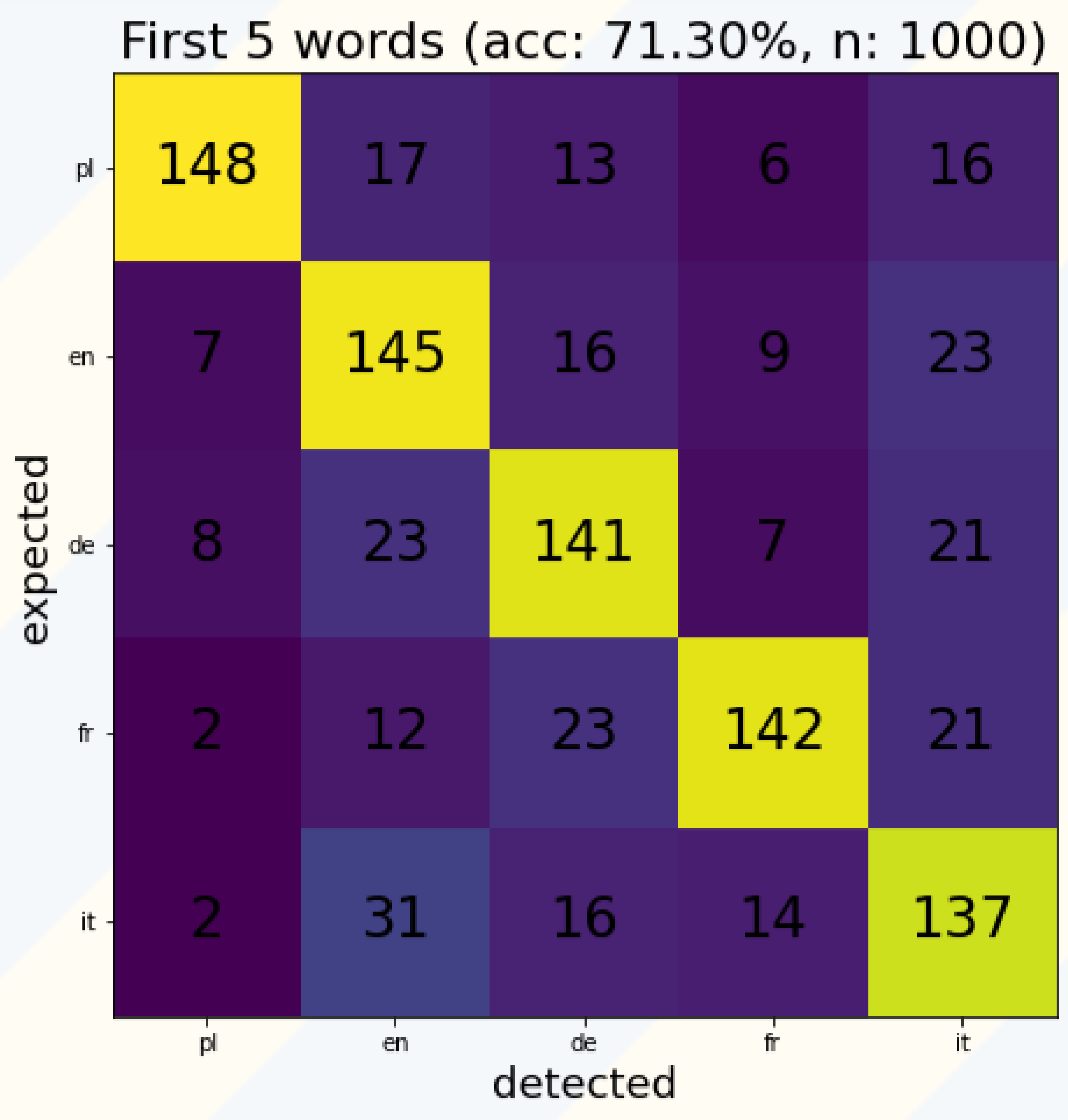
```
'en'
```

Evaluation on Sample Articles

```
show_evaluation(langdet, n=200)
```


Full Text (acc: 98.20%, n: 1000)

expected \ detected	pl	en	de	fr	it
pl	194	3	0	0	3
en	0	200	0	0	0
de	0	3	196	0	1
fr	0	1	0	198	1
it	0	5	1	0	194


First 5 words (acc: 71.30%, n: 1000)

expected \ detected	pl	en	de	fr	it
pl	148	17	13	6	16
en	7	145	16	9	23
de	8	23	141	7	21
fr	2	12	23	142	21
it	2	31	16	14	137

Pros and Cons

✓ Very simple idea and implementation

✓ Low memory requirement of the trained model (only n-gram rankings)

✓ Fast trainable (no gradients optimization)

✓ High tolerance for errors

✗ Poorly handles short and mixed texts

Don’t try this at work - Use nltk

The same n-gram based method has been implemented in nltk package.

```
!pip install nltk
```

Before the first run we must download additional language resources.

```
import nltk
nltk.download('crubadan');
```

Don’t try this at work - Use nltk

```python
from nltk.classify.textcat import TextCat
textcat = TextCat()

pl = textcat.guess_language("PyCon PL 2020 to trzynasta edycja "
    "ogólnopolskiej konferencji z grupy PyCon.")

en = textcat.guess_language("Capybaras are highly social, "
    "living in groups of up to 100 and communicating through "
    "a variety of vocalizations.")

(pl, en)
```

```
('pol', 'eng')
```

References

1. MediaWiki API Help.
2. Wikipedia contributors, "N-gram," Wikipedia, The Free Encyclopedia
3. Cavnar, W. B. and Trenkle, J. M. 1994. N-gram-based text categorization

Thank you for your attention :)

https://github.com/pkubiak/pl.pycon.2020