Testes de viabilidade

Caio Geraldes

14 de dezembro de 2021

Introdução

Este notebook testa a viabilidade dos algoritmos do módulo `lexicogenero`, de minha autoria, para selecionar, dividir e organizar dados de textos em prosa em grego antigo para a disciplina FLL5133-2021. O código fonte dos scripts utilizados aqui, bem como dos testes de qualidade estão disponíveis em meu GitHub.

1 Carregamento e organização dos dados importados do Diorisis

Pretendo utilizar o corpus anotado Diorisis de Alessandro Vatri e Barbara McGillyvray, disponível em formato `.json` e com lemmatização confiável (Vatri, 2020). Para processar esse corpus, utilizei as funções criadas em ../src/lexicogenero/ferramentas/diorisis_reader.py. A lista de *stop words* utilizada vem de Rodda, 2020, com algumas adições (código em ../src/lexicogenero/grc.py).

[1]:
```
import os
import itertools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.model_selection import train_test_split
```

[2]:
```
from lexicogenero.ferramentas.diorisis_reader import carrega_textos, em_pandas, sent_pandas
from lexicogenero.main import DIORISIS_PATH
from lexicogenero.grc import STOPS_LIST

%matplotlib inline
plt.rcParams['figure.figsize'] = [20, 5]
```

Neste teste, usaremos como corpus os textos de historiografia e prosa filosófica (diálogos e tratados) de autores do período clássico:

- Historiografia:
  - Heródoto
  - Xenofonte:

        * Ciropédia
        * Anábase
        * Helênica
    - Tucídides

- Prosa filosófica:

    - Platão
    - Xenofonte:
        * Agesilau
        * Hierão
        * Simpósio
        * Apologia
        * Memorabilia

[3]:
```
DATA = "data.csv"
SENTS = "sents.csv"
REAPROVEITAR = True

if not REAPROVEITAR or (DATA not in os.listdir() or SENTS not in os.listdir()):
    ignorados = [
        "Xenophon (0032) - On the Art of Horsemanship (013).json",
        "Xenophon (0032) - Economics (003).json",
        "Xenophon (0032) - Ways and Means (011).json",
        "Xenophon (0032) - Constitution of the Lacedaemonians (010).json",
        "Xenophon (0032) - On the Cavalry Commander (012).json",
        "Xenophon (0032) - On Hunting (014).json",
        "Xenophon (0032) - Apology (005).json",
    ]

    corpus = carrega_textos(autores=['Herodotus',
                                     'Thucydides',
                                     'Plato',
                                     'Xenophon (0032)', # Exclui Xenofonte de Éfeso
                                     ],
                            diorisis_path=DIORISIS_PATH,
                            ignore=ignorados,
                            verbose=False)
    df_tokens = em_pandas(corpus)
    df_sents = sent_pandas(corpus)
    del corpus
    df_tokens.to_csv(DATA, index=False)
    df_sents.to_csv(SENTS, index=False)

else:
    print('Carregando dataframe previamente salvo')
    df_tokens = pd.read_csv(DATA)
    df_sents = pd.read_csv(SENTS)
```

```
Carregando dataframe previamente salvo
```

Formato do DF: ainda não decidi se seria mais eficiente utilizar um data frame organizado por token ou por sentenças. A implementação de ambos é muito semelhante e pode ser vista em `../src/lexicogenero/ferramentas/diorisis_parser.py`

[4]:
```
df_tokens.head()
```

[4]:
```
   sent_id location           form          lemma      POS  \
0        1    1.t.1       Ἡροδότου       Ἡρόδοτος   proper
1        1    1.t.1  Ἁλικαρνησσέος  Ἁλικαρνησσεύς   proper
2        1    1.t.1       ἱστορίης        ἱστορία     noun
3        1    1.t.1       ἀπόδεξις       ἀπόδεξις     noun
4        1    1.t.1            ἥδε            ὅδε  pronoun

                  analyses  id                                    file  \
0              masc gen sg   1  Herodotus (0016) - Histories (001).json
1  masc gen sg (epic ionic)   2  Herodotus (0016) - Histories (001).json
2   fem gen sg (epic ionic)   3  Herodotus (0016) - Histories (001).json
3               fem nom sg   4  Herodotus (0016) - Histories (001).json
4               fem nom sg   5  Herodotus (0016) - Histories (001).json

      author       text
0  Herodotus  Histories
1  Herodotus  Histories
2  Herodotus  Histories
3  Herodotus  Histories
4  Herodotus  Histories
```

[5]:
```
df_sents.head()
```

[5]:
```
   sent_id location                                          forms  \
0        1    1.t.1  Ἡροδότου Ἁλικαρνησσέος ἱστορίης ἀπόδεξις ἥδε ,…
1        2    1.1.1  Περσέων μέν νυν οἱ λόγιοι Φοίνικας αἰτίους φασ…
2        3    1.1.2  τούτους γὰρ ἀπὸ τῆς Ἐρυθρῆς καλεομένης θαλάσση…
3        4    1.1.7  τὸ δὲ Ἄργος τοῦτον τὸν χρόνον προεῖχε ἅπασι τῶ…
4        5    1.1.9  ἀπικομένους δὲ τούς Φοίνικας ἐς δὴ τὸ Ἄργος το…

                                             lemmata  \
0  Ἡρόδοτος Ἁλικαρνησσεύς ἱστορία ἀπόδεξις ὅδε ὡς…
1  Πέρσευς μέν νῦν ὁ λόγιος Φοῖνιξ αἴτιος φημί γί…
2  οὗτος γάρ ἀπό ὁ Ἐρυθραί καλέω θάλασσα ἀφικνέομ…
3  ὁ δέ Ἄργος οὗτος ὁ χρόνος προέχω ἅπας ὁ ἐν ὁ ν…
4  ἀφικνέομαι δέ ὁ Φοῖνιξ εἰς δή ὁ Ἄργος οὗτος δι…

                                    file     author       text
0  Herodotus (0016) - Histories (001).json  Herodotus  Histories
1  Herodotus (0016) - Histories (001).json  Herodotus  Histories
2  Herodotus (0016) - Histories (001).json  Herodotus  Histories
```

```
3  Herodotus (0016) - Histories (001).json  Herodotus  Histories
4  Herodotus (0016) - Histories (001).json  Herodotus  Histories
```

1.1 Anotando Gênero

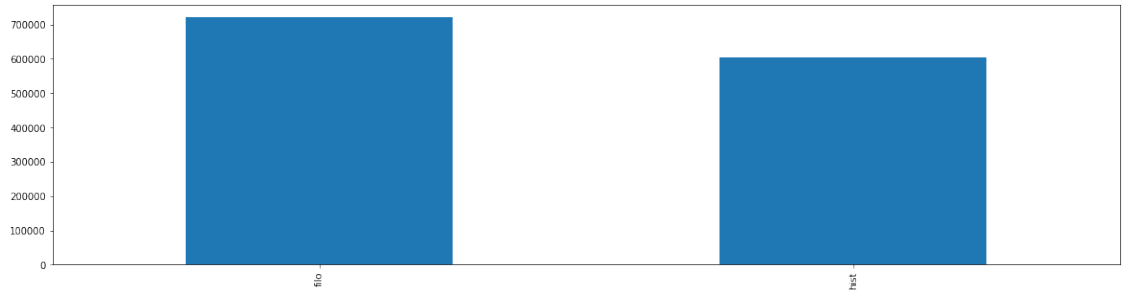
[6]:
```
hist = [
    "Herodotus (0016) - Histories (001).json",
    "Thucydides (0003) - History (001).json",
    "Xenophon (0032) - Hellenica (001).json",
    "Xenophon (0032) - Cyropaedia (007).json",
    "Xenophon (0032) - Anabasis (006).json",
]
```

[7]:
```
# Deve haver uma maneira menos rocambolesca para resolver isso
df_tokens['genero'] = np.nan
df_tokens.loc[df_tokens.file.isin(hist), 'genero'] = "hist"
df_tokens.genero.fillna("filo", inplace=True)
df_sents['genero'] = np.nan
df_sents.loc[df_sents.file.isin(hist), 'genero'] = "hist"
df_sents.genero.fillna("filo", inplace=True)
```

Para meu modelo funcionar, será necessário ter uma quantidade relativamente equilibrada de dados entre autores de prosa historiográfica e prosa filosófica, a seleção parece razoável. Qualquer tentativa de aumentar os dados de historiografia forçaria aumentar o escopo temporal (o próximo autor seria Políbio, já do período helenístico). Nota-se que a diferença aumenta quando tratamos de sentenças e não de tokens, indicando que as sentenças em textos historiográficos devem ser um pouco mais longas na média.

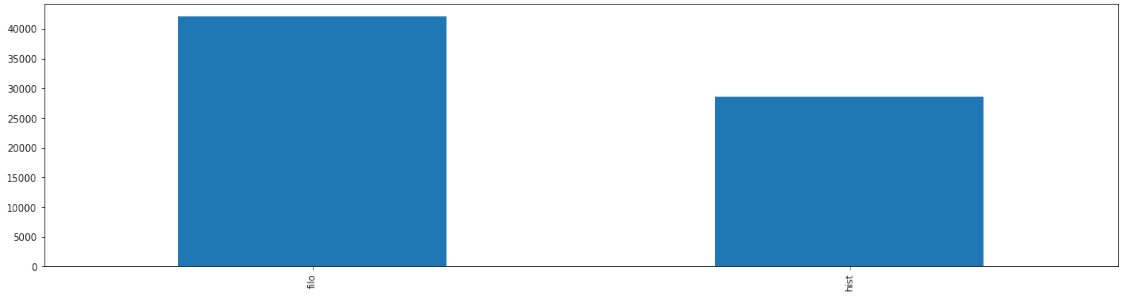
[8]:
```
df_tokens.genero.value_counts().plot(kind='bar')
```

[8]:
```
<AxesSubplot:>
```



[9]:
```
df_sents.genero.value_counts().plot(kind='bar')
```

[9]:
```
<AxesSubplot:>
```



1.2 Verbos por gênero (dataframe de tokens)

[10]:
```
df_verbos = df_tokens.loc[(df_tokens.POS == "verb") & (-df_tokens.lemma.isin(STOPS_LIST))]
df_verbos.head()
```

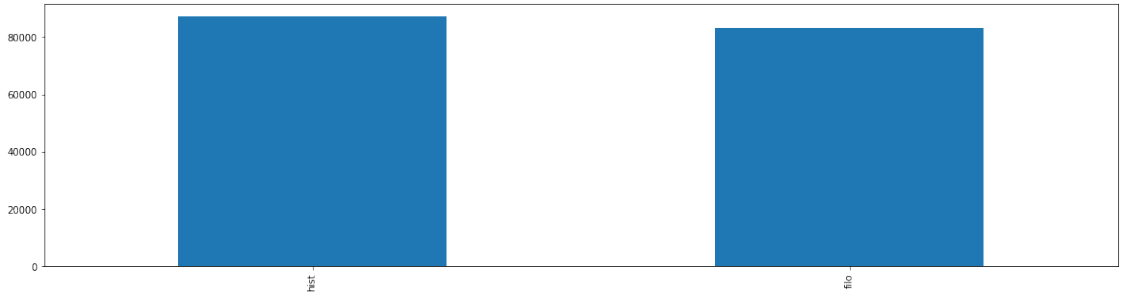
[10]:

	sent_id	location	form	lemma	POS	analyses	id	file	author	text	genero
30	1	1.t.1	ἀποδεχθέντα	ἀποδέχομαι	verb	aor part pass neut nom/voc/acc pl;aor part pas…	28	Herodotus (0016) - Histories (001).json	Herodotus	Histories	hist
42	1	1.t.1	ἐπολέμησαν	πολεμέω	verb	aor ind act 3rd pl	38	Herodotus (0016) - Histories (001).json	Herodotus	Histories	hist
62	3	1.1.2	καλεομένης	καλέω	verb	fut part mid fem gen sg (attic epic ionic);pre…	6	Herodotus (0016) - Histories (001).json	Herodotus	Histories	hist
64	3	1.1.2	ἀπικομένους	ἀφικνέομαι	verb	aor part mid masc acc pl (ionic)	8	Herodotus (0016) - Histories (001).json	Herodotus	Histories	hist
71	3	1.1.2	οἰκήσαντας	οἰκέω	verb	aor part act masc acc pl	14	Herodotus (0016) - Histories (001).json	Herodotus	Histories	hist

A filtragem não causa mudança na proporção entre documentos de historiografia e filosofia.

[11]:
```
df_verbos.genero.value_counts().plot(kind='bar')
```

[11]:
```
<AxesSubplot:>
```

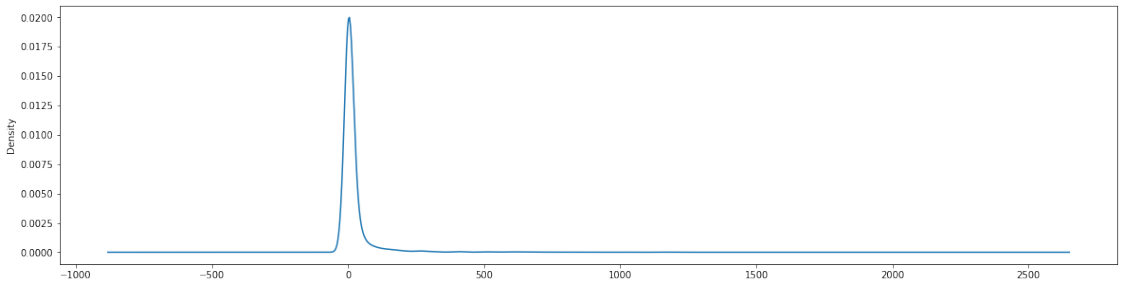


[12]:
```
filo_verbos = df_verbos[df_verbos.genero == "filo"].lemma
filo_verbos.value_counts()
```

[12]:
```
οἴομαι            1767
οἶδα              1482
φαίνω             1204
βούλομαι          1177
δεῖ               1026
                  …
προπαιδεύω           1
μολύνω               1
διαίνω               1
ὑπεραγανακτέω        1
ἐφέζομαι             1
Name: lemma, Length: 3663, dtype: int64
```

[13]:
```
filo_verbos.value_counts().plot(kind='density')
```

[13]:
```
<AxesSubplot:ylabel='Density'>
```



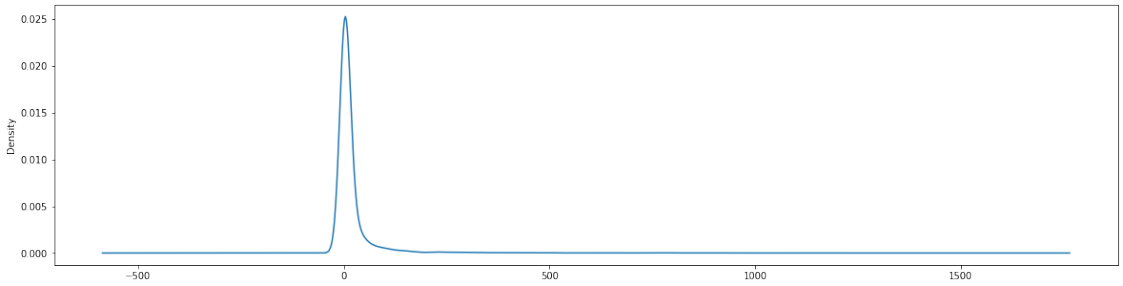
[14]:
```
hist_verbos = df_verbos[df_verbos.genero == "hist"].lemma
hist_verbos.value_counts()
```

[14]:
```
βούλομαι        1177
λαμβάνω          994
```

```
ἀφικνέομαι        915
ἔρχομαι           872
πάρειμι           822
                  …
ἐπεικάζω           1
ἀποκηδεύω          1
πυκνόω             1
καθιππάζομαι       1
ἐξαυλίζομαι        1
Name: lemma, Length: 4331, dtype: int64
```

[15]:
```
hist_verbos.value_counts().plot(kind='density')
```

[15]:
```
<AxesSubplot:ylabel='Density'>
```



1.3 Naive Bayes

Apenas de tokens

O resultado é terrível, provavelmente porque a implementação do modelo não é a melhor, afinal o DOCUMENTO e suas componentes são a mesma entidade.

[16]:
```
x, y = df_verbos.lemma, df_verbos.genero
```

[17]:
```
x_treino, x_teste, y_treino, y_teste = train_test_split(x, y, test_size=0.20,
↪random_state=42)

filo = [x for x, y in zip(x_treino, y_treino) if y == 'filo']
hist = [x for x, y in zip(x_treino, y_treino) if y == 'hist']
```

[18]:
```
vocab_filo = Counter(filo)
vocab_hist = Counter(hist)
```

[19]:
```
n_filo = len(filo)
n_hist = len(hist)
n = n_filo + n_hist
p_filo = np.log2(n_filo / n)
```

```
p_hist = np.log2(n_hist / n)
```

[20]:
```
n_vocab_filo = len(vocab_filo)
n_vocab_hist = len(vocab_hist)
```

[21]:
```
def bayes(texto):

    texto_filo = [x for x in texto if x in vocab_filo.keys()]
    c_doc_filo = Counter(texto_filo)
    texto_hist = [x for x in texto if x in vocab_hist.keys()]
    c_doc_hist = Counter(texto_hist)

    pf_filo = sum([np.log2((c_doc_filo[token] + 1) / n_vocab_filo + len(texto_filo)) for
 ↪token in texto_filo])
    pf_hist = sum([np.log2((c_doc_hist[token] + 1) / n_vocab_hist + len(texto_hist)) for
 ↪token in texto_hist])



    prob_filo = p_filo + pf_filo
    prob_hist = p_hist + pf_hist



    return prob_filo, prob_hist
```

[22]:
```
teste_pred_labels = []
for t in x_teste:
    prob_filo, prob_hist = bayes(t)
    if prob_filo >= prob_hist:
        teste_pred_labels.append("filo")
    else:
        teste_pred_labels.append("hist")
```

[23]:
```
teste_orig_labels = [x for x in y_teste]

performance_labels = []

for pred, label in zip(teste_pred_labels, teste_orig_labels):
    if label == "filo" and pred == "filo":
        performance_labels.append("VP")
    elif label == "filo" and pred == "hist":
        performance_labels.append("FP")
    elif label == "hist" and pred == "hist":
        performance_labels.append("VN")
    else:
        performance_labels.append("FN")

perf_counter = Counter(performance_labels)
```

[24]:
```
perf_counter
```

[24]:
```
Counter({'VN': 17407, 'FP': 16718})
```

Com sentenças

O modelo assim performa melhor, mas é pouco específico para os meus interesses: que o vocabulário da prosa filosófica e historiográfica são distintos diz pouco sobre sua seleção de verbos.

[25]:
```
def achatar(lista):
    return list(itertools.chain(*lista))

df_sents['lst_lemmata'] = df_sents.lemmata.str.split()
df_sents.dropna(inplace=True)
```

[26]:
```
x, y = df_sents.lst_lemmata, df_sents.genero

x_treino, x_teste, y_treino, y_teste = train_test_split(x, y, test_size=0.20,
↪random_state=42)
```

[27]:
```
filo = [x for x, y in zip(x_treino, y_treino) if y == 'filo']
hist = [x for x, y in zip(x_treino, y_treino) if y == 'hist']

vocab_filo = Counter([x for x in achatar(filo)])
vocab_hist = Counter([x for x in achatar(hist)])

n_filo = len(filo)
n_hist = len(hist)
n = n_filo + n_hist
p_filo = np.log2(n_filo / n)
p_hist = np.log2(n_hist / n)

n_vocab_filo = len(vocab_filo)
n_vocab_hist = len(vocab_hist)
```

[28]:
```
def bayes(texto):

    texto_filo = [x for x in texto if x in vocab_filo.keys()]
    c_doc_filo = Counter(texto_filo)
    texto_hist = [x for x in texto if x in vocab_hist.keys()]
    c_doc_hist = Counter(texto_hist)

    pf_filo = sum([np.log2((c_doc_filo[token] + 1) / n_vocab_filo + len(texto_filo)) for
↪token in texto_filo])
    pf_hist = sum([np.log2((c_doc_hist[token] + 1) / n_vocab_hist + len(texto_hist)) for
↪token in texto_hist])
```

```
    prob_filo = p_filo + pf_filo
    prob_hist = p_hist + pf_hist


    return prob_filo, prob_hist
```

[29]:
```
teste_pred_labels = []
for t in x_teste:
    prob_filo, prob_hist = bayes(t)
    if prob_filo >= prob_hist:
        teste_pred_labels.append("filo")
    else:
        teste_pred_labels.append("hist")

teste_orig_labels = [x for x in y_teste]
```

[30]:
```
performance_labels = []

for pred, label in zip(teste_pred_labels, teste_orig_labels):
    if label == "filo" and pred == "filo":
        performance_labels.append("VP")
    elif label == "filo" and pred == "hist":
        performance_labels.append("FP")
    elif label == "hist" and pred == "hist":
        performance_labels.append("VN")
    else:
        performance_labels.append("FN")
```

[31]:
```
perf_counter = Counter(performance_labels)

perf_counter
```

[31]:
```
Counter({'VP': 8254, 'VN': 2678, 'FN': 3046, 'FP': 165})
```

[32]:
```
precisao = perf_counter['VP'] / (perf_counter['VP'] + perf_counter['FP'])
cobertura = perf_counter['VP'] / (perf_counter['VP'] + perf_counter['FN'])
acuracia = (perf_counter['VP'] + perf_counter['VN']) / len(performance_labels)
medida_f = 2 * (precisao * cobertura) / (precisao + cobertura)

print(f'Precisão:  {precisao}')
print(f'Cobertura: {cobertura}')
print(f'Acurácia:  {acuracia}')
print(f'Medida F:  {medida_f}')
```

```
Precisão:  0.9804014728590094
Cobertura: 0.7304424778761062
Acurácia:  0.7729618892738457
Medida F:  0.8371621278969521
```

2 Conclusões

- O banco de dados anotado Diorisis oferece as anotações necessárias e é relativamente fácil de processar.
- O corpus proposto oferece um número relativamente equilibrado de dados para as categorias a serem classificadas e o algoritmo de processamento permite que ele seja expandido conforme necessário.
- Será importante revisar a suavização utilizada (no caso Laplace) para poder obter um modelo funcional se for utilizado um modelo como o Naive Bayes apenas com os verbos, embora a princípio já seja possível afirmar que esse modelo não funcionará bem assim.
- Se for possível codificar a informação de `df_sents.lst_lemmata` selecionar os verbos principais (utilizando o dependency parser do `stanza-perseus`), os resultados poderão ser mais consistentes com aqueles que são de meu interesse.