1 Tuning du modèle

L'objet de ce notebook est d'illustrer les différentes étapes de tuning du modèle.

1.1 Préambule

1.1.1 Imports

[1]:
```python
# setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)
```

[2]:
```python
# imports and customization of diplay
# import os
import re
from functools import partial
from itertools import product
import numpy as np
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width=108
# from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
# from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from matplotlib import pyplot as plt
import matplotlib.patches as mpatch
import matplotlib.ticker as mtick
import seaborn as sns

from src.pimest import ContentGetter
from src.pimest import PathGetter
from src.pimest import PDFContentParser
from src.pimest import BlockSplitter
from src.pimest import SimilaritySelector
from src.pimest import custom_accuracy
from src.pimest import text_sim_score
from src.pimest import text_similarity
# from src.pimest import build_text_processor
```

1.1.2 Acquisition des données

On récupère les données manuellement étiquetées et on les intègre dans un dataframe

[3]:
```python
ground_truth_df = pd.read_csv(Path('..') / '..' / 'ground_truth' / 'manually_labelled_ground_truth.csv',
                              sep=';',
                              encoding='latin-1',
                              index_col='uid')
ground_truth_uids = list(ground_truth_df.index)

acqui_pipe = Pipeline([('PathGetter', PathGetter(ground_truth_uids=ground_truth_uids,
                                                 train_set_path=Path('..') / '..' / 'ground_truth',
                                                 ground_truth_path=Path('..') / '..' / 'ground_truth',
                                                 )),
                        ('ContentGetter', ContentGetter(missing_file='to_nan')),
                        ('ContentParser', PDFContentParser(none_content='to_empty')),
                       ],
                       verbose=True)

texts_df = acqui_pipe.fit_transform(ground_truth_df)
texts_df['ingredients'] = texts_df['ingredients'].fillna('')
```

```
texts_df
```

```
[Pipeline] … (step 1 of 3) Processing PathGetter, total=   0.1s
[Pipeline] … (step 2 of 3) Processing ContentGetter, total=   0.6s
Launching 8 processes.
[Pipeline] … (step 3 of 3) Processing ContentParser, total=  37.0s
```

[3]:

```
                                                                           designation  \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70   Concentré liquide Asian en bouteille 980 ml CHEF
d183e914-db2f-4e2f-863a-a3b2d054c0b8             Pain burger curry 80 g CREATIV BURGER
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8                   Macaroni en sachet 500 g PANZANI
…                                                                                    …
e67341d8-350f-46f4-9154-4dbbb8035621              PRÉPARATION POUR CRÈME BRÛLÉE BIO 6L
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3  Céréales instantanées en poudre saveur caramel…
0faad739-ea8c-4f03-b62e-51ee592a0546                      FARINE DE BLÉ TYPE 45, 10KG

                                                                           ingredients  \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70  Eau, maltodextrine, sel, arômes, sucre, arôme …
d183e914-db2f-4e2f-863a-a3b2d054c0b8  Farine de blé T65, eau, levure, vinaigre de ci…
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8  - 100% Semoule de BLE dur de qualité supérieur…
…                                                                                    …
e67341d8-350f-46f4-9154-4dbbb8035621  Sucre roux de canne*° (64%), amidon de maïs*, …
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3  Farine 87,1 % (Blé (GLUTEN), Blé hydrolysé (GL…
0faad739-ea8c-4f03-b62e-51ee592a0546                                 Farine de blé T45

                                                                                  path  \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70  ../../ground_truth/a0492df6-9c76-4303-8813-65e…
d183e914-db2f-4e2f-863a-a3b2d054c0b8  ../../ground_truth/d183e914-db2f-4e2f-863a-a3b…
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8  ../../ground_truth/ab48a1ed-7a3d-4686-bb6d-ab4…
…                                                                                    …
e67341d8-350f-46f4-9154-4dbbb8035621  ../../ground_truth/e67341d8-350f-46f4-9154-4db…
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3  ../../ground_truth/a8f6f672-20ac-4ff8-a8f2-3bc…
0faad739-ea8c-4f03-b62e-51ee592a0546  ../../ground_truth/0faad739-ea8c-4f03-b62e-51e…

                                                                               content  \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70  b'%PDF-1.5\r\n%\xb5\xb5\xb5\xb5\r\n1 0 obj\r\n…
d183e914-db2f-4e2f-863a-a3b2d054c0b8  b'%PDF-1.5\r%\xe2\xe3\xcf\xd3\r\n4 0 obj\r<</L…
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8  b'%PDF-1.4\n%\xc7\xec\x8f\xa2\n5 0 obj\n<</Len…
…                                                                                    …
e67341d8-350f-46f4-9154-4dbbb8035621  b'%PDF-1.7\r\n%\xb5\xb5\xb5\xb5\r\n1 0 obj\r\n…
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3  b'%PDF-1.5\r\n%\xb5\xb5\xb5\xb5\r\n1 0 obj\r\n…
0faad739-ea8c-4f03-b62e-51ee592a0546  b'%PDF-1.5\r\n%\xb5\xb5\xb5\xb5\r\n1 0 obj\r\n…

                                                                                  text
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70  Concentré Liquide Asian CHEF® \n\nBouteille de…
d183e914-db2f-4e2f-863a-a3b2d054c0b8

ab48a1ed-7a3d-4686-bb6d-ab4f367cada8  Direction Qualité \n\n \n\n \n\nPATES ALIMENTA…
…                                                                                    …
e67341d8-350f-46f4-9154-4dbbb8035621  FICHE TECHNIQUE \n\nCREME BRÛLÉE 6L \n\nREF : …
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3  81 rue de Sans Souci - CS13754 - 69576 Limones…
0faad739-ea8c-4f03-b62e-51ee592a0546   \n1050/10502066400 \n\n10502055300/1050202520…

[500 rows x 5 columns]
```

[19]:

```
import inspect
```

[26]:

```
print(inspect.getsource(texts_df.to_string))
```

```
    @Substitution(
        header_type="bool or sequence",
```

```
    header="Write out the column names. If a list of strings "
    "is given, it is assumed to be aliases for the "
    "column names",
    col_space_type="int",
    col_space="The minimum width of each column",
)
@Substitution(shared_params=fmt.common_docstring, returns=fmt.return_docstring)
def to_string(
    self,
    buf: Optional[FilePathOrBuffer[str]] = None,
    columns: Optional[Sequence[str]] = None,
    col_space: Optional[int] = None,
    header: Union[bool, Sequence[str]] = True,
    index: bool = True,
    na_rep: str = "NaN",
    formatters: Optional[fmt.formatters_type] = None,
    float_format: Optional[fmt.float_format_type] = None,
    sparsify: Optional[bool] = None,
    index_names: bool = True,
    justify: Optional[str] = None,
    max_rows: Optional[int] = None,
    min_rows: Optional[int] = None,
    max_cols: Optional[int] = None,
    show_dimensions: bool = False,
    decimal: str = ".",
    line_width: Optional[int] = None,
    max_colwidth: Optional[int] = None,
    encoding: Optional[str] = None,
) -> Optional[str]:
    """
    Render a DataFrame to a console-friendly tabular output.
    %(shared_params)s
    line_width : int, optional
        Width to wrap a line in characters.
    max_colwidth : int, optional
        Max width to truncate each column in characters. By default, no limit.

        .. versionadded:: 1.0.0
    encoding : str, default "utf-8"
        Set character encoding.

        .. versionadded:: 1.0
    %(returns)s
    See Also
    --------
    to_html : Convert DataFrame to HTML.

    Examples
    --------
    >>> d = {'col1': [1, 2, 3], 'col2': [4, 5, 6]}
    >>> df = pd.DataFrame(d)
    >>> print(df.to_string())
       col1  col2
    0     1     4
    1     2     5
    2     3     6
    """

    from pandas import option_context

    with option_context("display.max_colwidth", max_colwidth):
        formatter = fmt.DataFrameFormatter(
            self,
            columns=columns,
            col_space=col_space,
            na_rep=na_rep,
            formatters=formatters,
            float_format=float_format,
```

```
            sparsify=sparsify,
            justify=justify,
            index_names=index_names,
            header=header,
            index=index,
            min_rows=min_rows,
            max_rows=max_rows,
            max_cols=max_cols,
            show_dimensions=show_dimensions,
            decimal=decimal,
            line_width=line_width,
        )
        return formatter.to_string(buf=buf, encoding=encoding)
```

1.1.3 Train / Test split

On va appliquer une grid search pour déterminer les meilleurs paramètres de notre modèle. Pour ne pas surestimer la performance du modèle, il est nécessaire de bien séparer le jeu de test du jeu d'entraînement, y compris pour la grid search !

[4]:
```
train, test = train_test_split(texts_df, test_size=100, random_state=42)
```

Dans toute la suite, on utilisera le jeu d'entraînement pour effectuer le tuning des hyperparamètres.

1.2 Ajustement de la fonction de découpage des textes

L'objectif de cette partie est d'optimiser la fonction de découpage des textes en blocs. On va tester quelques fonctions candidates, via une GridSearch.

1.2.1 Définition des fonctions candidates

On définit les fonctions de split :

[5]:
```
# definitions of splitter funcs
splitter_funcs = []
def split_func1(text):
    return(text.split('\n\n'))
splitter_funcs.append(split_func1)
def split_func2(text):
    return(text.split('\n'))
splitter_funcs.append(split_func2)
def split_func3(text):
    regex = r'\s*\n\s*\n\s*'
    return(re.split(regex, text))
splitter_funcs.append(split_func3)
```

1.2.2 Mise en place du pipeline

On construit ensuite un pipeline de traitement du texte. Le SimilaritySelector prenant en entrée une pandas.Series, on définit entre le BlockSplitter (dont la méthode transform() retourne un pandas.DataFrame) et le SimilaritySelector une fonction utilitaire qui séléctionne la colonne 'blocks'.

[6]:
```
def select_col(df, col_name='blocks'):
        return(df[col_name].fillna(''))
col_selector = FunctionTransformer(select_col)
```

[7]:
```
process_pipe = Pipeline([('Splitter', BlockSplitter()),
                         ('ColumnSelector', col_selector),
                         ('SimilaritySelector', SimilaritySelector())
                        ],
                       verbose=False)
```

On peut tester le fonctionnement de ce Pipeline. Attention, les résultats ne sont pas représentatifs, on entraîne et on prédit sur le même jeu de données !

[8]:
```
process_pipe.fit(train, train['ingredients'])
process_pipe.predict(train)
```

```
Launching 8 processes.
Launching 8 processes.
```

[8]:
```
uid
02d5ceb9-21c2-4965-8f65-309bca7638b2    Café chicorée solubles et fibres de chicorée.\…
bbe72396-6ed4-4df1-935b-0c0a7dbd77dc
507b428e-e99d-464b-b9d3-50629efe4355    COMPOSITION\nMélange de Blés de pays recommand…
                                                                    …
4b28bb17-1f1d-4cbb-ac3b-80227ef248ab    Gluten\nCrustacés\nOeufs\nPoisson\nSoja\nLait\…
d2137dae-ff21-46ec-83be-7400773c6c3b    Amidon modifié de pomme de terre - Fécule de p…
571d98ae-9647-4bd4-ad1a-a497f93987cb    Composition typique (Données inappropriées pou…
Length: 400, dtype: object
```

1.2.3 Helper fonction

On doit faire varier dans la grid search des paramètres qui sont packés sous forme de dictionnaires avant d'être passés au SimilaritySelector. On construit une fonction qui permet de construire le produit cartésien qui va bien pour ces paramètres.

[9]:
```
def prod_params(dict_to_prod):
    """
    In : dict of dicts.
    First level key : parameter name
    Second level key : name of scenario with this parameter value
    Values : parameter value

    Returns a tuple:
        - list of labels to name scenario
        - list of dictionaries to pass to count_vect_kwargs
    """
    label_lists = [list(dict_.keys()) for dict_ in dict_to_prod.values()]
    labels = list(map(lambda x: ', '.join(x), list(product(*label_lists))))
    values_iter = list(product(*[list(dict_.values()) for dict_ in dict_to_prod.values()]))
    parms_names = list(dict_to_prod.keys())
    dict_out = [{key: val for (key, val) in zip(parms_names, values_)} for values_ in values_iter]
    return(labels, dict_out)
```

[10]:
```
prod_params({'stop_words': {'no stopwords removal': None, 'with stopwords removal' : {'de', 'le'}},
             'ngram_ranges': {'no_ngram': (1, 1), 'bigrams': (1, 2)}})
```

[10]:
```
(['no stopwords removal, no_ngram',
  'no stopwords removal, bigrams',
  'with stopwords removal, no_ngram',
  'with stopwords removal, bigrams'],
 [{'stop_words': None, 'ngram_ranges': (1, 1)},
  {'stop_words': None, 'ngram_ranges': (1, 2)},
  {'stop_words': {'de', 'le'}, 'ngram_ranges': (1, 1)},
  {'stop_words': {'de', 'le'}, 'ngram_ranges': (1, 2)}])
```

1.2.4 Stockage des résultats dans un dataframe

Au fil des lancements des grid search, on stockera les données dans un dataframe afin de pouvoir les analyser plus simplement après coup.

[11]:
```
result_df = pd.DataFrame()
result_df
```

[11]:
```
Empty DataFrame
Columns: []
Index: []
```

1.2.5 Application de la GridSearch : tuning du text preprocessing (run 1)

On applique ensuite une grid search en faisant varier les fonctions de text preprocessing : - fonction de split du texte des documents en blocs - retrait ou non de stopwords - prise en compte de ngrams - juste pour une première comparaison, choix du candidat par projection l1/l2 ou par similarité cosinus

On scorera via la similarité de Levenshtein.

[12]:
```
lev_scorer = partial(text_sim_score, similarity='levenshtein')
```

[13]:
```
stop_words = {'pas', 'le', 'en', 'pour', 'ou', 'ce', 'de', 'dans', 'du', 'and', 'un', 'sur', 'et',
              'of', 'est', 'par', 'la', 'les', 'dont', 'au', 'des', 'que'}
```

[14]:
```
ngram_ranges = {'no_ngram': (1, 1), 'bigrams': (1, 2), 'trigrams': (1, 3)}
```

[15]:
```
kwargs_to_prod = prod_params({'stop_words': {'no stopwords removal': None, 'with stopwords removal' :␣
↪stop_words},
                              'ngram_range': ngram_ranges,
                              'strip_accents': {'keep accents': None, 'remove accents': 'unicode'}
                             }
                            )
```

[16]:
```
param_grid = [{'Splitter__splitter_func': splitter_funcs,
               'SimilaritySelector__similarity': ['projection', 'cosine'],
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
              }
             ]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                     ).fit(train, train['ingredients'])
```

```
Fitting 8 folds for each of 72 candidates, totalling 576 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   13.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  2.8min

Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 576 out of 576 | elapsed:  3.9min finished
```

[17]:
```
labels = list(product(kwargs_to_prod[0], ['Projection l2/l1', 'Cosinus'], ['Split 1', 'Split 2', 'Split␣
↪3']))
labels = list(map(lambda x: ', '.join(x), labels))
```

[18]:
```
for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.
↪cv_results_['std_test_similarity'][i]:.2%}"
    print(labels[i], str_result)
```

```
no stopwords removal, no_ngram, keep accents, Projection l2/l1, Split 1 50.15% +/- 5.61%
no stopwords removal, no_ngram, keep accents, Projection l2/l1, Split 2 38.90% +/- 4.52%
no stopwords removal, no_ngram, keep accents, Projection l2/l1, Split 3 52.65% +/- 6.09%
no stopwords removal, no_ngram, keep accents, Cosinus, Split 1 40.30% +/- 4.91%
no stopwords removal, no_ngram, keep accents, Cosinus, Split 2 25.87% +/- 2.25%
no stopwords removal, no_ngram, keep accents, Cosinus, Split 3 41.98% +/- 5.27%
no stopwords removal, no_ngram, remove accents, Projection l2/l1, Split 1 49.47% +/- 5.77%
no stopwords removal, no_ngram, remove accents, Projection l2/l1, Split 2 38.56% +/- 4.19%
no stopwords removal, no_ngram, remove accents, Projection l2/l1, Split 3 52.02% +/- 6.05%
```

```
no stopwords removal, no_ngram, remove accents, Cosinus, Split 1 40.93% +/- 5.02%
no stopwords removal, no_ngram, remove accents, Cosinus, Split 2 26.06% +/- 2.00%
no stopwords removal, no_ngram, remove accents, Cosinus, Split 3 42.68% +/- 5.42%
no stopwords removal, bigrams, keep accents, Projection l2/l1, Split 1 55.47% +/- 5.22%
no stopwords removal, bigrams, keep accents, Projection l2/l1, Split 2 43.12% +/- 2.32%
no stopwords removal, bigrams, keep accents, Projection l2/l1, Split 3 58.38% +/- 5.06%
no stopwords removal, bigrams, keep accents, Cosinus, Split 1 41.53% +/- 5.73%
no stopwords removal, bigrams, keep accents, Cosinus, Split 2 26.94% +/- 2.14%
no stopwords removal, bigrams, keep accents, Cosinus, Split 3 43.45% +/- 6.40%
no stopwords removal, bigrams, remove accents, Projection l2/l1, Split 1 55.40% +/- 5.11%
no stopwords removal, bigrams, remove accents, Projection l2/l1, Split 2 43.27% +/- 2.78%
no stopwords removal, bigrams, remove accents, Projection l2/l1, Split 3 58.31% +/- 5.23%
no stopwords removal, bigrams, remove accents, Cosinus, Split 1 41.49% +/- 5.88%
no stopwords removal, bigrams, remove accents, Cosinus, Split 2 27.27% +/- 1.95%
no stopwords removal, bigrams, remove accents, Cosinus, Split 3 43.70% +/- 6.48%
no stopwords removal, trigrams, keep accents, Projection l2/l1, Split 1 56.41% +/- 5.05%
no stopwords removal, trigrams, keep accents, Projection l2/l1, Split 2 43.75% +/- 3.06%
no stopwords removal, trigrams, keep accents, Projection l2/l1, Split 3 59.56% +/- 5.12%
no stopwords removal, trigrams, keep accents, Cosinus, Split 1 41.54% +/- 5.25%
no stopwords removal, trigrams, keep accents, Cosinus, Split 2 26.95% +/- 2.41%
no stopwords removal, trigrams, keep accents, Cosinus, Split 3 43.54% +/- 6.16%
no stopwords removal, trigrams, remove accents, Projection l2/l1, Split 1 56.43% +/- 5.07%
no stopwords removal, trigrams, remove accents, Projection l2/l1, Split 2 43.83% +/- 3.11%
no stopwords removal, trigrams, remove accents, Projection l2/l1, Split 3 59.58% +/- 5.11%
no stopwords removal, trigrams, remove accents, Cosinus, Split 1 42.08% +/- 5.14%
no stopwords removal, trigrams, remove accents, Cosinus, Split 2 27.28% +/- 2.39%
no stopwords removal, trigrams, remove accents, Cosinus, Split 3 44.18% +/- 6.17%
with stopwords removal, no_ngram, keep accents, Projection l2/l1, Split 1 54.94% +/- 5.62%
with stopwords removal, no_ngram, keep accents, Projection l2/l1, Split 2 42.12% +/- 4.32%
with stopwords removal, no_ngram, keep accents, Projection l2/l1, Split 3 58.06% +/- 6.52%
with stopwords removal, no_ngram, keep accents, Cosinus, Split 1 51.06% +/- 6.89%
with stopwords removal, no_ngram, keep accents, Cosinus, Split 2 29.73% +/- 4.76%
with stopwords removal, no_ngram, keep accents, Cosinus, Split 3 53.35% +/- 7.12%
with stopwords removal, no_ngram, remove accents, Projection l2/l1, Split 1 54.89% +/- 5.81%
with stopwords removal, no_ngram, remove accents, Projection l2/l1, Split 2 42.22% +/- 4.32%
with stopwords removal, no_ngram, remove accents, Projection l2/l1, Split 3 57.99% +/- 6.66%
with stopwords removal, no_ngram, remove accents, Cosinus, Split 1 51.40% +/- 6.72%
with stopwords removal, no_ngram, remove accents, Cosinus, Split 2 30.44% +/- 4.69%
with stopwords removal, no_ngram, remove accents, Cosinus, Split 3 53.69% +/- 7.13%
with stopwords removal, bigrams, keep accents, Projection l2/l1, Split 1 57.74% +/- 5.74%
with stopwords removal, bigrams, keep accents, Projection l2/l1, Split 2 44.54% +/- 3.57%
with stopwords removal, bigrams, keep accents, Projection l2/l1, Split 3 61.08% +/- 5.91%
with stopwords removal, bigrams, keep accents, Cosinus, Split 1 52.31% +/- 7.10%
with stopwords removal, bigrams, keep accents, Cosinus, Split 2 26.86% +/- 4.47%
with stopwords removal, bigrams, keep accents, Cosinus, Split 3 54.85% +/- 7.51%
with stopwords removal, bigrams, remove accents, Projection l2/l1, Split 1 57.99% +/- 5.69%
with stopwords removal, bigrams, remove accents, Projection l2/l1, Split 2 44.43% +/- 3.30%
with stopwords removal, bigrams, remove accents, Projection l2/l1, Split 3 60.86% +/- 5.65%
with stopwords removal, bigrams, remove accents, Cosinus, Split 1 51.79% +/- 6.83%
with stopwords removal, bigrams, remove accents, Cosinus, Split 2 26.90% +/- 4.71%
with stopwords removal, bigrams, remove accents, Cosinus, Split 3 54.43% +/- 7.18%
with stopwords removal, trigrams, keep accents, Projection l2/l1, Split 1 58.76% +/- 5.47%
with stopwords removal, trigrams, keep accents, Projection l2/l1, Split 2 45.36% +/- 3.58%
with stopwords removal, trigrams, keep accents, Projection l2/l1, Split 3 61.94% +/- 5.59%
with stopwords removal, trigrams, keep accents, Cosinus, Split 1 51.65% +/- 6.90%
with stopwords removal, trigrams, keep accents, Cosinus, Split 2 25.74% +/- 4.50%
with stopwords removal, trigrams, keep accents, Cosinus, Split 3 54.28% +/- 7.24%
with stopwords removal, trigrams, remove accents, Projection l2/l1, Split 1 58.82% +/- 5.48%
with stopwords removal, trigrams, remove accents, Projection l2/l1, Split 2 45.54% +/- 3.61%
with stopwords removal, trigrams, remove accents, Projection l2/l1, Split 3 62.01% +/- 5.61%
with stopwords removal, trigrams, remove accents, Cosinus, Split 1 51.54% +/- 6.74%
with stopwords removal, trigrams, remove accents, Cosinus, Split 2 26.04% +/- 4.42%
with stopwords removal, trigrams, remove accents, Cosinus, Split 3 54.26% +/- 7.21%
```

On tire de ce premier test: - que le modèle est bien plus performant avec le retrait des stopwords - que le split le plus efficace est la fonction qui applique la regex (deux retours chariots parmi des whitespaces) - split 3 - que la prise en compte de bigrammes améliore, avec les trigrammes en plus on ne gagne rien - que la similarité cosinus semble sensiblement moins performante que le choix par projection (l2/l1)

Remarque : la standard dev est quand même assez élevée (de l'ordre de 5-6%). Les scénarios avec peu d'écart entre leurs moyennes (2-3%) ne sont pas départageables via cette grid search.

On sauvegarde les résultat dans le dataframe qu'on analysera à la fin

[19]:
```
try:
    result_df = result_df.loc[result_df['run'] != 1].copy()
except:
    pass
result_df = pd.DataFrame(search.cv_results_)
result_df['run'] = 1
```

1.2.6 Application de la Grid Search : tuning du calcul de similarité (run 2)

On va maintenant déterminer, sur la base des paramètres déjà retenus, le mode de calcul de similarité le plus performant. Seul le calcul par projection est paramétrique (norme dans l'espace de départ vs. norme sur l'espace projeté), on fera uniquement varier ces paramètres (en plus de la comparaison avec la similarité cosinus).

On comparera également la performance du modèle selon qu'on vectorise les textes via les comptes de mots, ou bien seulement via un identifiant binaire (présence ou absence du mot).

[20]:
```
process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
                           })

kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words},
                              'ngram_range': {'bigrams': (1, 2)},
                              'binary': {'counts': False, 'binary flag': True},
                              'strip_accents': {'remove accents': 'unicode'}
                             })

param_grid = [{
               'SimilaritySelector__source_norm': ['l1'],
               'SimilaritySelector__projected_norm': ['l1'],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
              },
              {
               'SimilaritySelector__source_norm': ['l2'],
               'SimilaritySelector__projected_norm': ['l2'],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
              },
              {
               'SimilaritySelector__source_norm': ['l2'],
               'SimilaritySelector__projected_norm': ['l1'],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
              },
              {
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['l3'],
               'SimilaritySelector__projected_norm': ['l2'],
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
              },
              {
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['l4'],
               'SimilaritySelector__projected_norm': ['l3'],
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
              },
              {
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['l5'],
               'SimilaritySelector__projected_norm': ['l4'],
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
              },
              {
               'SimilaritySelector__similarity': ['projection'],
```

```
                'SimilaritySelector__source_norm': ['l6'],
                'SimilaritySelector__projected_norm': ['l5'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__source_norm': ['l3'],
                'SimilaritySelector__projected_norm': ['l1'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l4'],
                'SimilaritySelector__projected_norm': ['l2'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l5'],
                'SimilaritySelector__projected_norm': ['l3'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l6'],
                'SimilaritySelector__projected_norm': ['l4'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l7'],
                'SimilaritySelector__projected_norm': ['l5'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l4'],
                'SimilaritySelector__projected_norm': ['l1'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l5'],
                'SimilaritySelector__projected_norm': ['l2'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l6'],
                'SimilaritySelector__projected_norm': ['l3'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l7'],
                'SimilaritySelector__projected_norm': ['l4'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__source_norm': ['l8'],
                'SimilaritySelector__projected_norm': ['l5'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               },
               {
                'SimilaritySelector__similarity': ['cosine'],
                'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
```

```
            }
           ]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                     ).fit(train, train['ingredients'])
```

```
Fitting 8 folds for each of 36 candidates, totalling 288 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   14.7s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  1.2min

Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 288 out of 288 | elapsed:  1.8min finished
```

[21]:
```
labels = ['l1, l1',
          'l2, l2',
          'l2, l1',
          'l3, l2',
          'l4, l3',
          'l5, l4',
          'l6, l5',
          'l3, l1',
          'l4, l2',
          'l5, l3',
          'l6, l4',
          'l7, l5',
          'l4, l1',
          'l5, l2',
          'l6, l3',
          'l7, l4',
          'l8, l5',
          'cosine',
         ]

labels = list(product(labels, kwargs_to_prod[0]))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.
↪cv_results_['std_test_similarity'][i]:.2%}"
    print(labels[i], str_result)
```

```
l1, l1, with stopwords removal, bigrams, counts, remove accents 17.87% +/- 2.52%
l1, l1, with stopwords removal, bigrams, binary flag, remove accents 17.90% +/- 2.55%
l2, l2, with stopwords removal, bigrams, counts, remove accents 17.87% +/- 2.52%
l2, l2, with stopwords removal, bigrams, binary flag, remove accents 17.90% +/- 2.55%
l2, l1, with stopwords removal, bigrams, counts, remove accents 60.86% +/- 5.65%
l2, l1, with stopwords removal, bigrams, binary flag, remove accents 61.00% +/- 5.61%
l3, l2, with stopwords removal, bigrams, counts, remove accents 61.55% +/- 5.25%
l3, l2, with stopwords removal, bigrams, binary flag, remove accents 62.05% +/- 4.73%
l4, l3, with stopwords removal, bigrams, counts, remove accents 59.61% +/- 4.00%
l4, l3, with stopwords removal, bigrams, binary flag, remove accents 62.61% +/- 4.29%
l5, l4, with stopwords removal, bigrams, counts, remove accents 56.23% +/- 3.40%
l5, l4, with stopwords removal, bigrams, binary flag, remove accents 61.82% +/- 3.67%
l6, l5, with stopwords removal, bigrams, counts, remove accents 51.58% +/- 3.78%
l6, l5, with stopwords removal, bigrams, binary flag, remove accents 60.91% +/- 3.62%
l3, l1, with stopwords removal, bigrams, counts, remove accents 59.33% +/- 5.34%
l3, l1, with stopwords removal, bigrams, binary flag, remove accents 59.06% +/- 5.44%
l4, l2, with stopwords removal, bigrams, counts, remove accents 61.11% +/- 5.30%
l4, l2, with stopwords removal, bigrams, binary flag, remove accents 61.00% +/- 5.61%
```

```
l5, l3, with stopwords removal, bigrams, counts, remove accents 59.28% +/- 3.60%
l5, l3, with stopwords removal, bigrams, binary flag, remove accents 61.70% +/- 5.21%
l6, l4, with stopwords removal, bigrams, counts, remove accents 55.73% +/- 4.26%
l6, l4, with stopwords removal, bigrams, binary flag, remove accents 62.05% +/- 4.73%
l7, l5, with stopwords removal, bigrams, counts, remove accents 50.18% +/- 2.42%
l7, l5, with stopwords removal, bigrams, binary flag, remove accents 61.96% +/- 4.23%
l4, l1, with stopwords removal, bigrams, counts, remove accents 58.42% +/- 5.53%
l4, l1, with stopwords removal, bigrams, binary flag, remove accents 57.44% +/- 5.05%
l5, l2, with stopwords removal, bigrams, counts, remove accents 60.29% +/- 4.59%
l5, l2, with stopwords removal, bigrams, binary flag, remove accents 59.67% +/- 5.18%
l6, l3, with stopwords removal, bigrams, counts, remove accents 58.12% +/- 3.67%
l6, l3, with stopwords removal, bigrams, binary flag, remove accents 61.00% +/- 5.61%
l7, l4, with stopwords removal, bigrams, counts, remove accents 53.31% +/- 2.96%
l7, l4, with stopwords removal, bigrams, binary flag, remove accents 61.50% +/- 5.35%
l8, l5, with stopwords removal, bigrams, counts, remove accents 47.87% +/- 2.99%
l8, l5, with stopwords removal, bigrams, binary flag, remove accents 61.80% +/- 5.20%
cosine, with stopwords removal, bigrams, counts, remove accents 54.43% +/- 7.18%
cosine, with stopwords removal, bigrams, binary flag, remove accents 53.36% +/- 7.86%
```

On tire de ce second test les conclusions suivantes : - comme lors du premier test, l'identification du meilleur candidat par similarité cosinus est moins performante que par projection - plusieurs configurations de paramètres permettent d'obtenir des performance similaires via la projection : - l2/l1 - l2/l1b - l3/l2 - l3/l2b - l3/l1b - l4/l2 - l4/l2b

[22]:
```python
result_df = result_df.loc[result_df['run'] != 2.].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(2)
len(result_df)
```

[22]:
```
108
```

1.2.7 Application de la Grid Search : impact des mots non vus en entrainement (run 3)

On va également voir si l'utilisation d'un vectorizer de type HashingVectorizer, qui permet de prendre en compte des mots non vus lors de l'entraînement a un impact sur la performance (ou son écart type, qui est très élevé…).

[23]:
```python
process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],

                           })

kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words},
                              'ngram_range': {'bigrams': (1, 2)},
                              'binary': {'counts': False, 'binary flag': True},
                             })

param_grid = [{'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__count_vect_type': ['TfidfVectorizer', 'HashingVectorizer'],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['l4'],
               'SimilaritySelector__projected_norm': ['l2'],
              },
              {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__count_vect_type': ['TfidfVectorizer', 'HashingVectorizer'],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['l3'],
               'SimilaritySelector__projected_norm': ['l2'],
              },
              {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__count_vect_type': ['TfidfVectorizer', 'HashingVectorizer'],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['l2'],
               'SimilaritySelector__projected_norm': ['l1'],
              },
              {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__count_vect_type': ['TfidfVectorizer', 'HashingVectorizer'],
               'SimilaritySelector__similarity': ['cosine'],
              },
             ]
```

```
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                     ).fit(train, train['ingredients'])
```

```
Fitting 8 folds for each of 16 candidates, totalling 128 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   13.7s

Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 128 out of 128 | elapsed:   52.8s finished
```

[24]:
```
labels = [
          'l4/l2',
          'l3/l2',
          'l2/l1',
          'cosine'
         ]

labels = list(product(labels, kwargs_to_prod[0], ['TfidfVectorizer', 'HashingVectorizer']))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.
↪cv_results_['std_test_similarity'][i]:.2%}"
    print(labels[i], str_result)
```

```
l4/l2, with stopwords removal, bigrams, counts, TfidfVectorizer 61.11% +/- 5.30%
l4/l2, with stopwords removal, bigrams, counts, HashingVectorizer 52.46% +/- 3.06%
l4/l2, with stopwords removal, bigrams, binary flag, TfidfVectorizer 61.00% +/- 5.61%
l4/l2, with stopwords removal, bigrams, binary flag, HashingVectorizer 54.13% +/- 4.12%
l3/l2, with stopwords removal, bigrams, counts, TfidfVectorizer 61.55% +/- 5.25%
l3/l2, with stopwords removal, bigrams, counts, HashingVectorizer 43.37% +/- 4.64%
l3/l2, with stopwords removal, bigrams, binary flag, TfidfVectorizer 62.05% +/- 4.73%
l3/l2, with stopwords removal, bigrams, binary flag, HashingVectorizer 47.26% +/- 5.17%
l2/l1, with stopwords removal, bigrams, counts, TfidfVectorizer 60.86% +/- 5.65%
l2/l1, with stopwords removal, bigrams, counts, HashingVectorizer 57.89% +/- 5.60%
l2/l1, with stopwords removal, bigrams, binary flag, TfidfVectorizer 61.00% +/- 5.61%
l2/l1, with stopwords removal, bigrams, binary flag, HashingVectorizer 58.26% +/- 5.01%
cosine, with stopwords removal, bigrams, counts, TfidfVectorizer 54.43% +/- 7.18%
cosine, with stopwords removal, bigrams, counts, HashingVectorizer 53.01% +/- 6.20%
cosine, with stopwords removal, bigrams, binary flag, TfidfVectorizer 53.36% +/- 7.86%
cosine, with stopwords removal, bigrams, binary flag, HashingVectorizer 50.56% +/- 7.20%
```

L'utilisation d'un HashingVectorizer à la place d'un TfidfVectorizer, pour prendre en compte les mots non vus lors de l'entrainement, n'a pas d'impact positif sur la performance du modèle. Au contraire, elle semble globalement diminuer de quelques points.

[25]:
```
result_df = result_df.loc[result_df['run'] != 3.].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(3)
len(result_df)
```

[25]: 124

1.2.8 Application d'une grid search : pondération des mots (run 4)

On va en plus appliquer une pondération absolue et relative des mots, dans la recherche de similarité par cosinus.

Les différentes possibilités pour le vecteur cible sont : - moyenne des vecteurs de textes des listes d'ingrédients, avec uniquement un flag binaire (présence / absence du mot) : la cible est la document frequency moyenne des mots des listes d'ingrédients - moyenne des vecteurs de textes des listes d'ingrédients, avec en prenant en compte les comptes des mots dans chacun des textes : la cible est la term frequency moyenne des mots au sein des listes d'ingrédients - moyenne des scores "absolus" de chacun des mots au sein des listes d'ingrédients. Il s'agit d'une "smooth document frequency" (elle croit logarithmiquement) - moyenne des scores "relatifs" de chacun des mots entre liste d'ingrédients et contenu des fiches techniques. Ici on compare la doc frequency entre les deux corpus, pour donner plus de poids aux mots qui sont plus présents dans les listes d'ingrédients que dans le reste du corps du texte.

On comparera à la projection l4/l2b, qui porte jusque là les meilleurs résultats.

[26]:
```
process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
                           'SimilaritySelector__count_vect_type': 'TfidfVectorizer',
                           })

kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words},
                              'ngram_range': {'no_ngrams': (1, 1), 'bigrams': (1, 2)},
                              'binary': {'counts': False, 'binary flag': True},
                              'use_idf': {'without idf': False, 'with idf': True},
                              'strip_accents': {'remove accents': 'unicode'},
                             })

param_grid = [{
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__scoring': ['default', 'absolute_score', 'relative_score'],
               'SimilaritySelector__similarity': ['cosine'],
              },
              {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['l4'],
               'SimilaritySelector__projected_norm': ['l2'],
              },
             ]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                     ).fit(train, train['ingredients'])
```

```
Fitting 8 folds for each of 32 candidates, totalling 256 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   15.0s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  1.4min

Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 256 out of 256 | elapsed:  1.8min finished
```

[27]:
```
labels = [
          'cosine',
         ]


labels = list(product(labels, kwargs_to_prod[0], ['default', 'absolute score', 'relative score']))
labels.extend(list(product(['projection l4/l2'],  kwargs_to_prod[0])))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.
↪cv_results_['std_test_similarity'][i]:.2%}"
    print(labels[i], str_result)
```

```
cosine, with stopwords removal, no_ngrams, counts, without idf, remove accents, default 53.69% +/- 7.13%
cosine, with stopwords removal, no_ngrams, counts, without idf, remove accents, absolute score 54.47% +/-
6.87%
cosine, with stopwords removal, no_ngrams, counts, without idf, remove accents, relative score 29.42% +/-
5.32%
cosine, with stopwords removal, no_ngrams, counts, with idf, remove accents, default 55.47% +/- 6.70%
cosine, with stopwords removal, no_ngrams, counts, with idf, remove accents, absolute score 52.47% +/- 6.91%
cosine, with stopwords removal, no_ngrams, counts, with idf, remove accents, relative score 32.73% +/- 5.21%
cosine, with stopwords removal, no_ngrams, binary flag, without idf, remove accents, default 53.29% +/-
7.86%
cosine, with stopwords removal, no_ngrams, binary flag, without idf, remove accents, absolute score 54.15%
+/- 8.17%
cosine, with stopwords removal, no_ngrams, binary flag, without idf, remove accents, relative score 28.02%
+/- 5.00%
cosine, with stopwords removal, no_ngrams, binary flag, with idf, remove accents, default 54.91% +/- 7.31%
cosine, with stopwords removal, no_ngrams, binary flag, with idf, remove accents, absolute score 51.80% +/-
8.13%
cosine, with stopwords removal, no_ngrams, binary flag, with idf, remove accents, relative score 31.39% +/-
5.59%
cosine, with stopwords removal, bigrams, counts, without idf, remove accents, default 54.43% +/- 7.18%
cosine, with stopwords removal, bigrams, counts, without idf, remove accents, absolute score 55.48% +/-
7.11%
cosine, with stopwords removal, bigrams, counts, without idf, remove accents, relative score 33.39% +/-
6.11%
cosine, with stopwords removal, bigrams, counts, with idf, remove accents, default 55.86% +/- 6.78%
cosine, with stopwords removal, bigrams, counts, with idf, remove accents, absolute score 52.00% +/- 6.91%
cosine, with stopwords removal, bigrams, counts, with idf, remove accents, relative score 39.00% +/- 5.38%
cosine, with stopwords removal, bigrams, binary flag, without idf, remove accents, default 53.36% +/- 7.86%
cosine, with stopwords removal, bigrams, binary flag, without idf, remove accents, absolute score 55.36% +/-
7.36%
cosine, with stopwords removal, bigrams, binary flag, without idf, remove accents, relative score 32.74% +/-
5.87%
cosine, with stopwords removal, bigrams, binary flag, with idf, remove accents, default 54.73% +/- 7.39%
cosine, with stopwords removal, bigrams, binary flag, with idf, remove accents, absolute score 51.12% +/-
6.71%
cosine, with stopwords removal, bigrams, binary flag, with idf, remove accents, relative score 39.45% +/-
5.47%
projection l4/l2, with stopwords removal, no_ngrams, counts, without idf, remove accents 57.08% +/- 5.38%
projection l4/l2, with stopwords removal, no_ngrams, counts, with idf, remove accents 17.38% +/- 1.45%
projection l4/l2, with stopwords removal, no_ngrams, binary flag, without idf, remove accents 57.26% +/-
5.89%
projection l4/l2, with stopwords removal, no_ngrams, binary flag, with idf, remove accents 22.50% +/- 4.14%
projection l4/l2, with stopwords removal, bigrams, counts, without idf, remove accents 61.11% +/- 5.30%
projection l4/l2, with stopwords removal, bigrams, counts, with idf, remove accents 32.00% +/- 3.51%
projection l4/l2, with stopwords removal, bigrams, binary flag, without idf, remove accents 61.00% +/- 5.61%
projection l4/l2, with stopwords removal, bigrams, binary flag, with idf, remove accents 38.00% +/- 5.85%
```

[28]:
```
result_df = result_df.loc[result_df['run'] != 4.].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(4)
len(result_df)
```

[28]:
```
156
```

On en déduit : - que la similarité par projection reste le mode de détermination du candidat le plus efficace - que dans ce mode, l'utilisation de l'idf dégrade la performance - néanmoins, dans le cadre de la similarité cosinus, l'utilisation de l'idf a un impact positif pour la fonction de scoring par défaut, ou relative.

1.2.9 Application de la grid search : embeddings des mots (run 5)

On mesure l'impact sur la performance de l'utilisation d'embeddings de mots.

[29]:
```
process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
                           'SimilaritySelector__count_vect_type': 'TfidfVectorizer',
                           })

kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words},
```

```
                                'ngram_range': {'no_ngram': (1, 1)},
                                'binary': {'counts': False, 'binary flag': True},
                                'use_idf': {'without_idf': False, 'with_idf': True},
                                'strip_accents': {'remove accents': 'unicode'},
                               })

param_grid = [{
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__scoring': ['default', 'absolute_score', 'relative_score'],
               'SimilaritySelector__similarity': ['cosine'],
               'SimilaritySelector__embedding_method': [None, 'Word2Vec', 'tSVD'],
              },
             ]
search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                      error_score='raise',
                     ).fit(train, train['ingredients'])
```

```
Fitting 8 folds for each of 36 candidates, totalling 288 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   26.8s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  2.3min

Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 288 out of 288 | elapsed:  3.7min finished
```

[30]:
```
labels = ['default', 'absolute_score', 'relative_score']


labels = list(product(kwargs_to_prod[0],
                      ['No embed', 'Word2Vec', 'tSVD'],
                      ['default', 'absolute score', 'relative score'],
                     ))
# labels.extend(list(product(['projection l4/l2'],  kwargs_to_prod[0])))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.
↪cv_results_['std_test_similarity'][i]:.2%}"
    print(labels[i], str_result)
```

```
with stopwords removal, no_ngram, counts, without_idf, remove accents, No embed, default 53.69% +/- 7.13%
with stopwords removal, no_ngram, counts, without_idf, remove accents, No embed, absolute score 54.47% +/-
6.87%
with stopwords removal, no_ngram, counts, without_idf, remove accents, No embed, relative score 29.42% +/-
5.32%
with stopwords removal, no_ngram, counts, without_idf, remove accents, Word2Vec, default 53.32% +/- 5.99%
with stopwords removal, no_ngram, counts, without_idf, remove accents, Word2Vec, absolute score 52.95% +/-
5.50%
with stopwords removal, no_ngram, counts, without_idf, remove accents, Word2Vec, relative score 10.21% +/-
2.68%
with stopwords removal, no_ngram, counts, without_idf, remove accents, tSVD, default 51.56% +/- 7.38%
with stopwords removal, no_ngram, counts, without_idf, remove accents, tSVD, absolute score 49.75% +/- 7.89%
with stopwords removal, no_ngram, counts, without_idf, remove accents, tSVD, relative score 7.94% +/- 1.56%
with stopwords removal, no_ngram, counts, with_idf, remove accents, No embed, default 55.47% +/- 6.70%
with stopwords removal, no_ngram, counts, with_idf, remove accents, No embed, absolute score 52.47% +/-
6.91%
with stopwords removal, no_ngram, counts, with_idf, remove accents, No embed, relative score 32.73% +/-
5.21%
with stopwords removal, no_ngram, counts, with_idf, remove accents, Word2Vec, default 53.39% +/- 5.76%
```

```
with stopwords removal, no_ngram, counts, with_idf, remove accents, Word2Vec, absolute score 53.66% +/-
6.19%
with stopwords removal, no_ngram, counts, with_idf, remove accents, Word2Vec, relative score 10.17% +/-
2.66%
with stopwords removal, no_ngram, counts, with_idf, remove accents, tSVD, default 48.99% +/- 6.04%
with stopwords removal, no_ngram, counts, with_idf, remove accents, tSVD, absolute score 45.60% +/- 5.59%
with stopwords removal, no_ngram, counts, with_idf, remove accents, tSVD, relative score 9.15% +/- 2.08%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, No embed, default 53.29% +/-
7.86%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, No embed, absolute score 54.15%
+/- 8.17%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, No embed, relative score 28.02%
+/- 5.00%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, Word2Vec, default 52.09% +/-
5.94%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, Word2Vec, absolute score 52.89%
+/- 6.00%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, Word2Vec, relative score 10.18%
+/- 2.67%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, tSVD, default 54.65% +/- 8.83%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, tSVD, absolute score 54.34% +/-
8.17%
with stopwords removal, no_ngram, binary flag, without_idf, remove accents, tSVD, relative score 9.57% +/-
2.68%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, No embed, default 54.91% +/- 7.31%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, No embed, absolute score 51.80% +/-
8.13%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, No embed, relative score 31.39% +/-
5.59%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, Word2Vec, default 53.77% +/- 6.61%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, Word2Vec, absolute score 53.28% +/-
5.81%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, Word2Vec, relative score 10.17% +/-
2.66%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, tSVD, default 50.53% +/- 7.48%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, tSVD, absolute score 49.09% +/-
7.51%
with stopwords removal, no_ngram, binary flag, with_idf, remove accents, tSVD, relative score 9.03% +/-
3.18%
```

[31]:
```python
result_df = result_df.loc[result_df['run'] != 5].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(5)
len(result_df)
```

[31]: 192

1.2.10 Random search : validation finale de l'ensemble des critères (run 6)

On applique enfin une random search, afin de voir si les conclusions qui avaient été tirée lors des explorations systématiques de certains domaines sont viables.

[32]:
```python
kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words, 'keep stopwords': None},
                              'use_idf': {'with idf': True, 'no idf': False},
                              'binary': {'counts': False, 'binary flag': True},
                              'ngram_range': {'no_ngram': (1, 1), 'bigrams': (1, 2), 'trigrams': (1, 3)},
                              'strip_accents': {'remove accents': 'unicode', 'keep accents': None},
                             })
param_grid = [{
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__scoring': ['default', 'absolute_score', 'relative_score'],
               'SimilaritySelector__similarity': ['cosine'],
               'SimilaritySelector__embedding_method': [None, 'Word2Vec', 'tSVD'],

              },
              {'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
```

```python
                'SimilaritySelector__scoring': ['default'],
                'SimilaritySelector__similarity': ['projection'],
                'SimilaritySelector__source_norm': ['l2', 'l3', 'l4', 'l5'],
                'SimilaritySelector__projected_norm': ['l1', 'l2', 'l3', 'l4'],
               },
              ]
len(kwargs_to_prod[1])

search = RandomizedSearchCV(process_pipe,
                            param_grid,
                            n_iter=50,
                            cv=8,
                            scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                            refit='similarity',
                            n_jobs=-1,
                            verbose=1,
                           ).fit(train, train['ingredients'])
```

```
Fitting 8 folds for each of 50 candidates, totalling 400 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   14.5s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  2.0min

Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 400 out of 400 | elapsed:  5.8min finished
```

[33]:
```python
search.best_params_
```

[33]:
```
{'SimilaritySelector__source_norm': 'l2',
 'SimilaritySelector__similarity': 'projection',
 'SimilaritySelector__scoring': 'default',
 'SimilaritySelector__projected_norm': 'l1',
 'SimilaritySelector__count_vect_kwargs': {'stop_words': {'and',
   'au',
   'ce',
   'dans',
   'de',
   'des',
   'dont',
   'du',
   'en',
   'est',
   'et',
   'la',
   'le',
   'les',
   'of',
   'ou',
   'par',
   'pas',
   'pour',
   'que',
   'sur',
   'un'},
  'use_idf': False,
  'binary': True,
  'ngram_range': (1, 3),
  'strip_accents': 'unicode'}}
```

[34]:
```python
search.best_score_
```

[34]:
```
0.619700702176325
```

1.2.11 Tuning des méthodes de vectorisation (run 7)

[35]:
```
process_pipe.set_params(**{'Splitter__splitter_func': splitter_funcs[2],
                          })

kwargs_to_prod = prod_params({'stop_words': {'with stopwords removal' : stop_words},
                              'use_idf': {'with idf': True, 'no idf': False},
                              'binary': {'counts': False, 'binary flag': True},
                              'ngram_range': {'no_ngram': (1, 1),
                                              'bigrams': (1, 2),
                                              'trigrams': (1, 3),
                                              'quadgrams': (1, 4),
                                              'quintgrams': (1, 5)},
                              'strip_accents': {'remove accents': 'unicode'},
                             })

param_grid = [{
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__similarity': ['cosine'],
              },
              {
               'SimilaritySelector__count_vect_kwargs': kwargs_to_prod[1],
               'SimilaritySelector__similarity': ['projection'],
               'SimilaritySelector__source_norm': ['l4'],
               'SimilaritySelector__projected_norm': ['l3'],
              }]

search = GridSearchCV(process_pipe,
                      param_grid,
                      cv=8,
                      scoring= ({'similarity': lev_scorer, 'accuracy': custom_accuracy}),
                      refit='similarity',
                      n_jobs=-1,
                      verbose=1,
                      error_score='raise',
                     ).fit(train, train['ingredients'])
```

```
Fitting 8 folds for each of 40 candidates, totalling 320 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   18.9s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  1.5min

Launching 8 processes.

[Parallel(n_jobs=-1)]: Done 320 out of 320 | elapsed:  2.5min finished
```

[36]:
```
labels = list(product(['cosine', 'Proj l4/l3'],
                      kwargs_to_prod[0],
                     ))
# labels.extend(list(product(['projection l4/l2'],  kwargs_to_prod[0])))
labels = list(map(lambda x: ', '.join(x), labels))

for i in range(len(search.cv_results_['rank_test_similarity'])):
    str_result = f"{search.cv_results_['mean_test_similarity'][i]:.2%} +/- {search.
↪cv_results_['std_test_similarity'][i]:.2%}"
    print(labels[i], str_result)
```

```
cosine, with stopwords removal, with idf, counts, no_ngram, remove accents 55.47% +/- 6.70%
cosine, with stopwords removal, with idf, counts, bigrams, remove accents 55.86% +/- 6.78%
cosine, with stopwords removal, with idf, counts, trigrams, remove accents 55.01% +/- 6.50%
cosine, with stopwords removal, with idf, counts, quadgrams, remove accents 55.44% +/- 6.40%
cosine, with stopwords removal, with idf, counts, quintgrams, remove accents 55.62% +/- 6.47%
cosine, with stopwords removal, with idf, binary flag, no_ngram, remove accents 54.91% +/- 7.31%
cosine, with stopwords removal, with idf, binary flag, bigrams, remove accents 54.73% +/- 7.39%
cosine, with stopwords removal, with idf, binary flag, trigrams, remove accents 54.68% +/- 7.98%
cosine, with stopwords removal, with idf, binary flag, quadgrams, remove accents 54.23% +/- 7.59%
```

```
cosine, with stopwords removal, with idf, binary flag, quintgrams, remove accents 54.10% +/- 8.03%
cosine, with stopwords removal, no idf, counts, no_ngram, remove accents 53.69% +/- 7.13%
cosine, with stopwords removal, no idf, counts, bigrams, remove accents 54.43% +/- 7.18%
cosine, with stopwords removal, no idf, counts, trigrams, remove accents 54.26% +/- 7.21%
cosine, with stopwords removal, no idf, counts, quadgrams, remove accents 54.12% +/- 7.28%
cosine, with stopwords removal, no idf, counts, quintgrams, remove accents 54.29% +/- 7.21%
cosine, with stopwords removal, no idf, binary flag, no_ngram, remove accents 53.29% +/- 7.86%
cosine, with stopwords removal, no idf, binary flag, bigrams, remove accents 53.36% +/- 7.86%
cosine, with stopwords removal, no idf, binary flag, trigrams, remove accents 52.39% +/- 7.54%
cosine, with stopwords removal, no idf, binary flag, quadgrams, remove accents 51.73% +/- 7.89%
cosine, with stopwords removal, no idf, binary flag, quintgrams, remove accents 51.04% +/- 8.52%
Proj 14/13, with stopwords removal, with idf, counts, no_ngram, remove accents 10.23% +/- 1.89%
Proj 14/13, with stopwords removal, with idf, counts, bigrams, remove accents 9.05% +/- 2.18%
Proj 14/13, with stopwords removal, with idf, counts, trigrams, remove accents 8.95% +/- 2.33%
Proj 14/13, with stopwords removal, with idf, counts, quadgrams, remove accents 8.97% +/- 2.30%
Proj 14/13, with stopwords removal, with idf, counts, quintgrams, remove accents 8.97% +/- 2.30%
Proj 14/13, with stopwords removal, with idf, binary flag, no_ngram, remove accents 9.50% +/- 1.73%
Proj 14/13, with stopwords removal, with idf, binary flag, bigrams, remove accents 9.28% +/- 2.34%
Proj 14/13, with stopwords removal, with idf, binary flag, trigrams, remove accents 9.49% +/- 2.21%
Proj 14/13, with stopwords removal, with idf, binary flag, quadgrams, remove accents 9.65% +/- 2.34%
Proj 14/13, with stopwords removal, with idf, binary flag, quintgrams, remove accents 9.65% +/- 2.34%
Proj 14/13, with stopwords removal, no idf, counts, no_ngram, remove accents 56.36% +/- 4.14%
Proj 14/13, with stopwords removal, no idf, counts, bigrams, remove accents 59.61% +/- 4.00%
Proj 14/13, with stopwords removal, no idf, counts, trigrams, remove accents 60.39% +/- 3.35%
Proj 14/13, with stopwords removal, no idf, counts, quadgrams, remove accents 60.72% +/- 3.33%
Proj 14/13, with stopwords removal, no idf, counts, quintgrams, remove accents 60.87% +/- 3.19%
Proj 14/13, with stopwords removal, no idf, binary flag, no_ngram, remove accents 59.16% +/- 6.49%
Proj 14/13, with stopwords removal, no idf, binary flag, bigrams, remove accents 62.61% +/- 4.29%
Proj 14/13, with stopwords removal, no idf, binary flag, trigrams, remove accents 63.31% +/- 3.83%
Proj 14/13, with stopwords removal, no idf, binary flag, quadgrams, remove accents 63.21% +/- 3.84%
Proj 14/13, with stopwords removal, no idf, binary flag, quintgrams, remove accents 63.21% +/- 3.83%
```

[37]:
```
result_df = result_df.loc[result_df['run'] != 7.].copy()
result_df = pd.concat([result_df, pd.DataFrame(search.cv_results_)], axis=0, ignore_index=True)
result_df['run'] = result_df['run'].fillna(7)
len(result_df)
```

[37]: 232

1.2.12 Dépouillement des résultats

Préparation du dataframe On va maintenant interpréter le contenu du dataframe portant les résultats. On commence par renommer les colonnes qui ont de longs noms.

[38]:
```
col_rename = {
              'param_SimilaritySelector__similarity': 'similarity',
              'param_Splitter__splitter_func': 'split_func',
              'param_SimilaritySelector__projected_norm': 'projected_norm',
              'param_SimilaritySelector__source_norm': 'source_norm',
              'param_SimilaritySelector__count_vect_type': 'count_vect_type',
              'param_SimilaritySelector__scoring': 'scoring',
              'param_SimilaritySelector__embedding_method': 'embedding_method',
}
result_df.rename(col_rename, axis=1, inplace=True)
```

On récupère maintenant le contenu des dictionnaires en tant que colonnes.

[39]:
```
dict_cols = {'param_SimilaritySelector__count_vect_kwargs'}
to_concat = list()
for dict_col in dict_cols:
    to_concat.append(result_df[dict_col].apply(pd.Series))
    result_df.drop(dict_col, axis=1, inplace=True)
result_df = pd.concat([result_df, *to_concat], axis=1)
```

[40]:
```
result_df.sample(3)
```

[40]:

```
     mean_fit_time  std_fit_time  mean_score_time  std_score_time  similarity split_func  \
151       0.814467      0.064605         0.761567        0.066706  projection        NaN
219       1.647092      0.087426         1.187234        0.084328  projection        NaN
140       1.774916      0.102340         1.383115        0.128081      cosine        NaN

                                               params  split0_test_similarity  split1_test_similarity  \
151  {'SimilaritySelector__count_vect_kwargs': {'st…                0.229513                0.266554
219  {'SimilaritySelector__count_vect_kwargs': {'st…                0.114523                0.085755
140  {'SimilaritySelector__count_vect_kwargs': {'st…                0.457836                0.623919

     split2_test_similarity  …  projected_norm  source_norm  count_vect_type         scoring  \
151                0.227790  …              12           14              NaN             NaN
219                0.128418  …              13           14              NaN             NaN
140                0.496019  …             NaN          NaN              NaN  absolute_score

     embedding_method                                         stop_words  ngram_range  strip_accents  \
151               NaN  {du, ce, par, de, et, est, la, des, sur, ou, d…       (1, 1)        unicode
219               NaN  {du, ce, par, de, et, est, la, des, sur, ou, d…       (1, 3)        unicode
140               NaN  {du, ce, par, de, et, est, la, des, sur, ou, d…       (1, 2)        unicode

     binary  use_idf
151    True     True
219    True     True
140   False     True

[3 rows x 40 columns]
```

On effectue ensuite quelques prétraitements pour améliorer la lisibilité. On renomme les fonctions avec leur nom, et on applique une valeur par défaut.

[41]:

```python
def rename_funcs(func):
    try:
        return(func.__name__)
    except:
        return('split_func3')
result_df['split_func'] = result_df.loc[:, 'split_func'].apply(rename_funcs)
```

On met 2 critères pour le retrait des stopwords : retrait d'une liste, ou non retrait.

[42]:

```python
result_df['stop_words'].fillna('no_stopword_removal', inplace=True)
result_df.loc[result_df['stop_words'] != 'no_stopword_removal', 'stop_words'] = 'stop_word_list'
```

On renomme les ngram_ranges

[43]:

```python
ngram_dict = {(1, 1): 'no_ngram',
              (1, 2): 'bigrams',
              (1, 3): 'trigrams',
              (1, 4): 'quadgrams',
              (1, 5): 'quintgrams',
             }
result_df['ngram_range'] = result_df['ngram_range'].map(ngram_dict, na_action='ignore')
```

On renomme le retrait des accents.

[44]:

```python
accent_dict = {None: 'no_accent_removal',
               'unicode': 'accent_removal',
              }
result_df['strip_accents'] = result_df['strip_accents'].map(accent_dict, na_action='ignore')
```

[45]:

```python
result_df.drop('params', axis=1, inplace=True)
```

On renomme les embeddings

[46]:

```python
embed_dict = {None: 'no_embedding',
              'Word2Vec': 'Word2Vec',
              'tSVD': 'tSVD',
```

```
                }
result_df['embedding_method'] = result_df['embedding_method'].map(embed_dict, na_action='ignore')
```

On applique ensuite les valeurs par défaut pour les colonnes sur lesquelles on n'a pas fait varier les critères.

[47]:
```
# by default, accents are stripped
result_df['strip_accents'] = result_df['strip_accents'].fillna('accent_removal')
# scoring and embedding method are not applicable if projection
result_df.loc[result_df['similarity'] == 'projection',['scoring', 'embedding_method']] = 'not_applicable'
# add default value to scoring for remainder
result_df.loc[pd.isna(result_df['scoring']), 'scoring'] = 'default'
# default value for embedding for remainder
result_df['embedding_method'] = result_df['embedding_method'].fillna('no_embedding')
# projected and source norm are not applicable if similarity is cosine
result_df.loc[result_df['similarity'] == 'cosine',['source_norm', 'projected_norm']] = 'not_applicable'
# default value for norms for remainder
result_df['source_norm'] = result_df['source_norm'].fillna('l2')
result_df['projected_norm'] = result_df['projected_norm'].fillna('l1')
```

[48]:
```
fillna_dict = {
                'similarity': 'projection',
                'split_func': 'split_func3',
                'count_vect_type': 'TfidfVectorizer',
                'use_idf': False,
                'binary': False,
              }
for key, val in fillna_dict.items():
    result_df[key] = result_df[key].fillna(val)
```

[49]:
```
parms = ['split_func',
         'stop_words',
         'strip_accents',
         'binary',
         'use_idf',
         'ngram_range',
         'similarity',
         'projected_norm',
         'source_norm',
         'count_vect_type',
         'scoring',
         'embedding_method',
        ]
result_df.reset_index().set_index(parms).drop('index', axis=1).sort_index()
```

[49]:
```
                                                          mean_fit_time  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding          1.351985
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable        1.272452
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding          0.858885
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
TfidfVectorizer not_applicable not_applicable        0.679841
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding          1.396822
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable        1.398119

                                                          std_fit_time  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
```

```
not_applicable TfidfVectorizer default        no_embedding          0.122508
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable       0.103519
                                                                  no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding          0.061593
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
TfidfVectorizer not_applicable not_applicable       0.043345
                                                                  trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding          0.111692
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable       0.130913

                                                            mean_score_time  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding             1.624542
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable          0.821700
                                                                  no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding             0.809040
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
TfidfVectorizer not_applicable not_applicable          0.570729
                                                                  trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding             1.147467
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable          0.751918

                                                            std_score_time  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding            0.318567
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable         0.084806
                                                                  no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding            0.040443
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
TfidfVectorizer not_applicable not_applicable         0.058958
                                                                  trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding            0.119107
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable         0.079647

                                                            split0_test_similarity  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                   0.383472
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                0.545019
                                                                  no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                   0.380675
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
TfidfVectorizer not_applicable not_applicable                0.542261
                                                                  trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                   0.444685
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                0.565570
```

```
                                                           split1_test_similarity  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring       embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.500577
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.621710
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.503208
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.653969
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.647609
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.692604

                                                           split2_test_similarity  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring       embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.324671
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.451395
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.334112
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.484972
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.478009
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.553719

                                                           split3_test_similarity  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring       embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.433144
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.557046
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.410601
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.573402
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.560348
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.593826

                                                           split4_test_similarity  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring       embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.464408
                                                                             projection l1             l2
TfidfVectorizer not_applicable not_applicable                 0.605446
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default       no_embedding                    0.440279
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1             l2
```

```
TfidfVectorizer not_applicable not_applicable                 0.702138
                                                                trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                  0.662707
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                 0.716421

                                                          split5_test_similarity  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                  0.367341
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                 0.526797
                                                                no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                  0.385640
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                 0.563652
                                                                trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                  0.518518
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                 0.596413

                                                          …  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method  …
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding      …
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable    …
                                                                no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding      …
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable    …
                                                                trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding      …
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable    …

                                                          split2_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                    0.12
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                   0.18
                                                                no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                    0.12
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                   0.16
                                                                trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                    0.20
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                   0.24

                                                          split3_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                    0.16
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                   0.16
```

```
                                                                  no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.16
…
…
split_func3 stop_word_list      no_accent_removal False  False    no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.18
                                                                  trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.20
                                                                              projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.20

                                                            split4_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False    bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.14
                                                                              projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.20
                                                                  no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.12
…
…
split_func3 stop_word_list      no_accent_removal False  False    no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.24
                                                                  trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.22
                                                                              projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.26

                                                            split5_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False    bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.18
                                                                              projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.26
                                                                  no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.20
…
…
split_func3 stop_word_list      no_accent_removal False  False    no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.26
                                                                  trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.20
                                                                              projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.28

                                                            split6_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False    bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.12
                                                                              projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.22
                                                                  no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.12
…
…
split_func3 stop_word_list      no_accent_removal False  False    no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.22
                                                                  trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                       0.14
                                                                              projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.24

                                                            split7_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
```

```
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                    0.14
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.18
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                    0.14
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.18
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                    0.14
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                     0.18

                                                           mean_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                0.1450
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                 0.2075
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                0.1450
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                 0.2075
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                0.1900
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                 0.2350

                                                           std_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding             0.019365
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable              0.031524
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding             0.025981
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable              0.031524
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding             0.038730
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable              0.029580

                                                           rank_test_accuracy  \
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                   43
                                                                            projection l1            l2
TfidfVectorizer not_applicable not_applicable                    16
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                   43
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable                    16
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding                   27
```

```
                                                                          projection l1            l2
TfidfVectorizer not_applicable not_applicable                     1


                                                           run
split_func  stop_words          strip_accents     binary use_idf ngram_range similarity projected_norm
source_norm    count_vect_type scoring        embedding_method
split_func1 no_stopword_removal accent_removal    False  False   bigrams     cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding      1.0
                                                                          projection l1            l2
TfidfVectorizer not_applicable not_applicable    1.0
                                                                 no_ngram    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding      1.0
…
…
split_func3 stop_word_list      no_accent_removal False  False   no_ngram    projection l1            l2
TfidfVectorizer not_applicable not_applicable    1.0
                                                                 trigrams    cosine     not_applicable
not_applicable TfidfVectorizer default        no_embedding      1.0
                                                                          projection l1            l2
TfidfVectorizer not_applicable not_applicable    1.0

[232 rows x 27 columns]
```

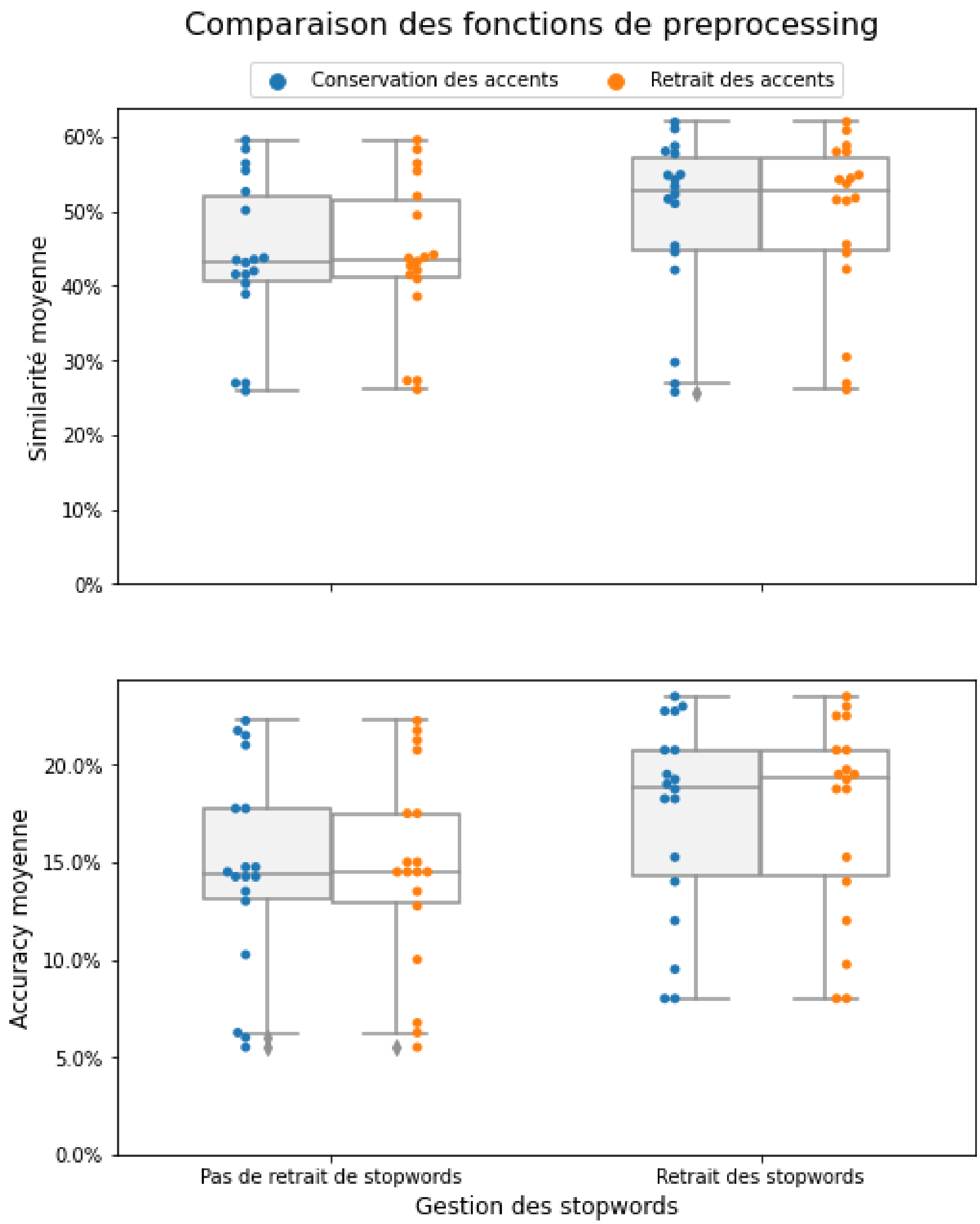
[50]:
```
result_df.to_csv(Path('.') / 'model_tuning_results.csv')
```

[51]:
```
result_df = pd.read_csv(Path('.') / 'model_tuning_results.csv')
```

Sélection des fonctions de preprocessing

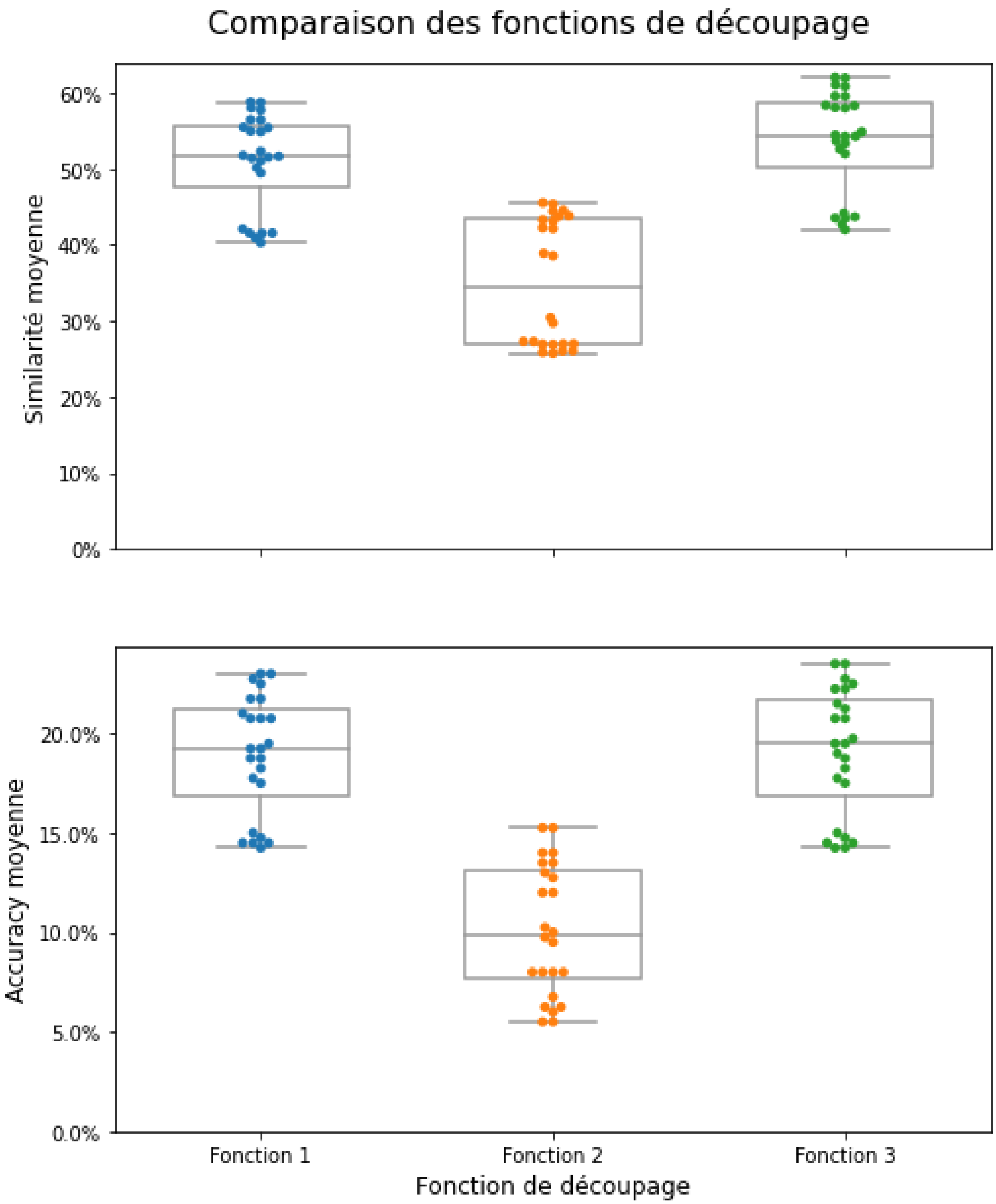
[52]:
```
fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[result_df['run'] == 1],
         'x': 'stop_words',
         'hue': 'strip_accents',
        }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
        #                   color='blue',
                          ax=axs[i],
                         )

    sns.boxplot(**parms,
                y=feature,
                ax=axs[i],
                color='white',
                width=.6,
                )

    axs[i].set_ylim(0,)
    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    axs[i].get_legend().remove()
    axs[i].set_xlabel('')

axs[1].set_xlabel('Gestion des stopwords', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Pas de retrait de stopwords', 'Retrait des stopwords'])

fig.legend(handles=axs[0].get_legend_handles_labels()[0][2:],
           labels=['Conservation des accents', 'Retrait des accents'],
           loc='center',
```

```
            ncol=2,
            bbox_to_anchor=(0, 1, 1, 0.12),
            bbox_transform=axs[0].transAxes,
           )

fig.suptitle('Comparaison des fonctions de preprocessing', fontsize=16, y=.95)

fig.savefig(Path('..') / 'img' / 'tuning_prepro.png', bbox_inches='tight')
```



Sélection de la fonction de split

[53]:
```
fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[result_df['run'] == 1],
         'x': 'split_func',
         'hue': None,
        }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
        #                   color='blue',
                          ax=axs[i],
                         )

    sns.boxplot(**parms,
                y=feature,
                ax=axs[i],
                color='white',
                width=.6,
                )

    axs[i].set_ylim(0,)
    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
#     axs[i].get_legend().remove()
    axs[i].set_xlabel('')

axs[1].set_xlabel('Fonction de découpage', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Fonction 1', 'Fonction 2', 'Fonction 3'])

# fig.legend(handles=axs[0].get_legend_handles_labels()[0][2:],
#            labels=['Conservation des accents', 'Retrait des accents'],
#            loc='center',
#            ncol=2,
#            bbox_to_anchor=(0, 1, 1, 0.12),
#            bbox_transform=axs[0].transAxes,
#           )

fig.suptitle('Comparaison des fonctions de découpage', fontsize=16, y=.92)

fig.savefig(Path('..') / 'img' / 'tuning_split.png', bbox_inches='tight')
```



Comparatif des similarités

[54]:
```
result_df['simil_kind'] = result_df['similarity'] + result_df['source_norm'] + result_df['projected_norm']
```

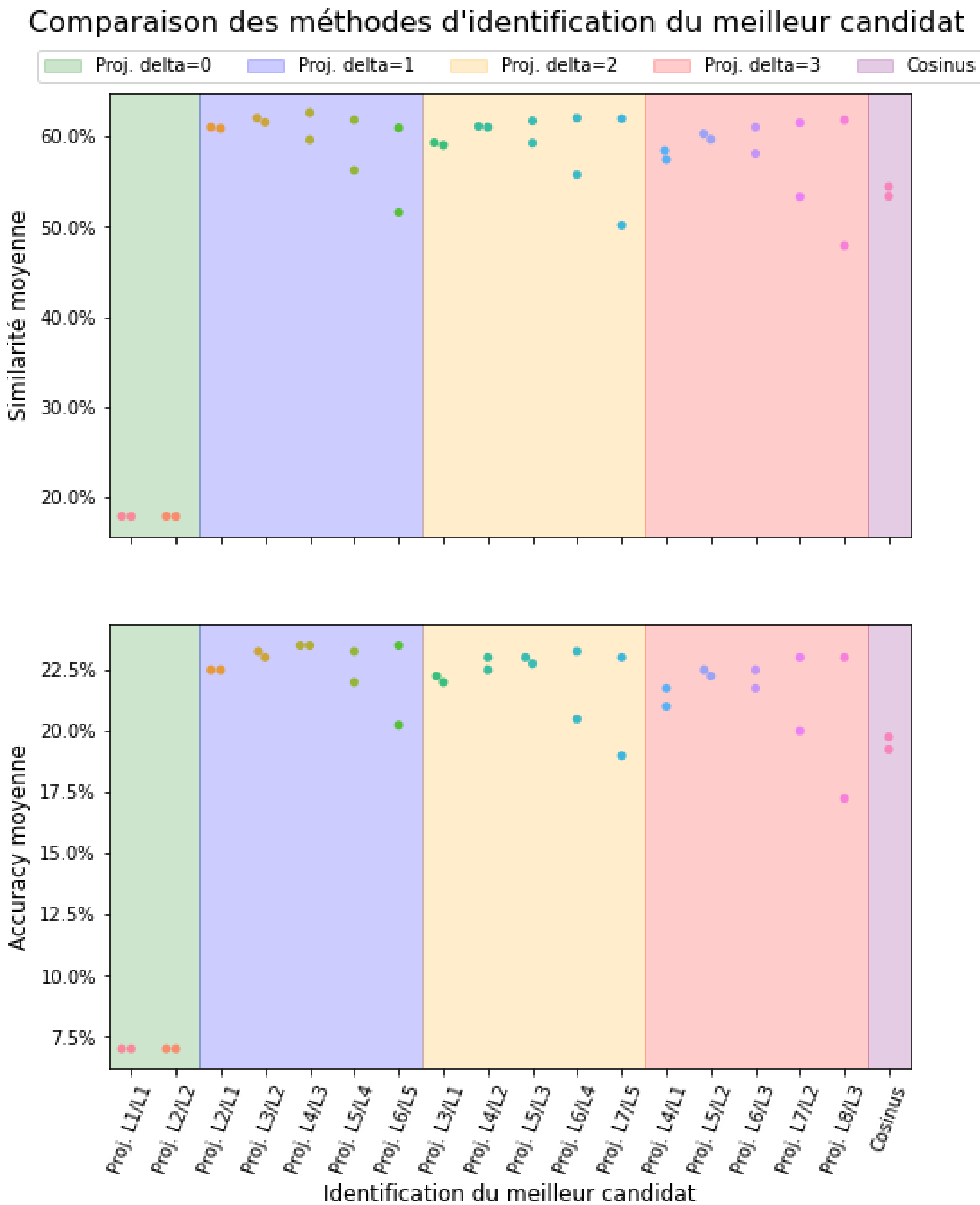
[55]:
```
result_df['embedding_method'].unique()
```

[55]:
```
array(['not_applicable', 'no_embedding', 'Word2Vec', 'tSVD'], dtype=object)
```

[56]:

```python
fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[(result_df['run'] == 2) &
                               (result_df['strip_accents'] == 'accent_removal') &
                               (result_df['stop_words'] == 'stop_word_list') &
                               (result_df['scoring'].isin({'default', 'not_applicable'})) &
                               (result_df['embedding_method'].isin({'no_embedding', 'not_applicable'}))],
         'x': 'simil_kind',
         'hue': None,
        }

patch_list = [
              mpatch.Rectangle((-1, 0), 2.5, 1, color='green', alpha=.2, edgecolor=None),
              mpatch.Rectangle((1.5, 0), 5, 1, color='blue', alpha=.2, edgecolor=None),
              mpatch.Rectangle((6.5, 0), 5, 1, color='orange', alpha=.2, edgecolor=None),
              mpatch.Rectangle((11.5, 0), 5, 1, color='red', alpha=.2, edgecolor=None),
              mpatch.Rectangle((16.5, 0), 2, 1, color='purple', alpha=.2, edgecolor=None),
              mpatch.Rectangle((-1, 0), 2.5, 1, color='green', alpha=.2, edgecolor=None),
              mpatch.Rectangle((1.5, 0), 5, 1, color='blue', alpha=.2, edgecolor=None),
              mpatch.Rectangle((6.5, 0), 5, 1, color='orange', alpha=.2, edgecolor=None),
              mpatch.Rectangle((11.5, 0), 5, 1, color='red', alpha=.2, edgecolor=None),
              mpatch.Rectangle((16.5, 0), 2, 1, color='purple', alpha=.2, edgecolor=None),
             ]

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
        #                   color='blue',
                          ax=axs[i],
                         )

    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    axs[i].set_xlabel('')
    for j in range(len(patch_list) // 2):
        axs[i].add_patch(patch_list[i * len(patch_list) // 2 + j])

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
labels = ['Proj. L1/L1',
          'Proj. L2/L2',
          'Proj. L2/L1',
          'Proj. L3/L2',
          'Proj. L4/L3',
          'Proj. L5/L4',
          'Proj. L6/L5',
          'Proj. L3/L1',
          'Proj. L4/L2',
          'Proj. L5/L3',
          'Proj. L6/L4',
          'Proj. L7/L5',
          'Proj. L4/L1',
          'Proj. L5/L2',
          'Proj. L6/L3',
          'Proj. L7/L2',
          'Proj. L8/L3',
          'Cosinus']
plt.setp(axs[1].xaxis.get_majorticklabels(), rotation=70)
axs[1].set_xticklabels(labels)
fig.legend(handles=patch_list[len(patch_list) // 2:],
           labels=['Proj. delta=0', 'Proj. delta=1', 'Proj. delta=2', 'Proj. delta=3', 'Cosinus'],
           loc='center',
           ncol=5,
           bbox_to_anchor=(0, 1, 1, 0.12),
```

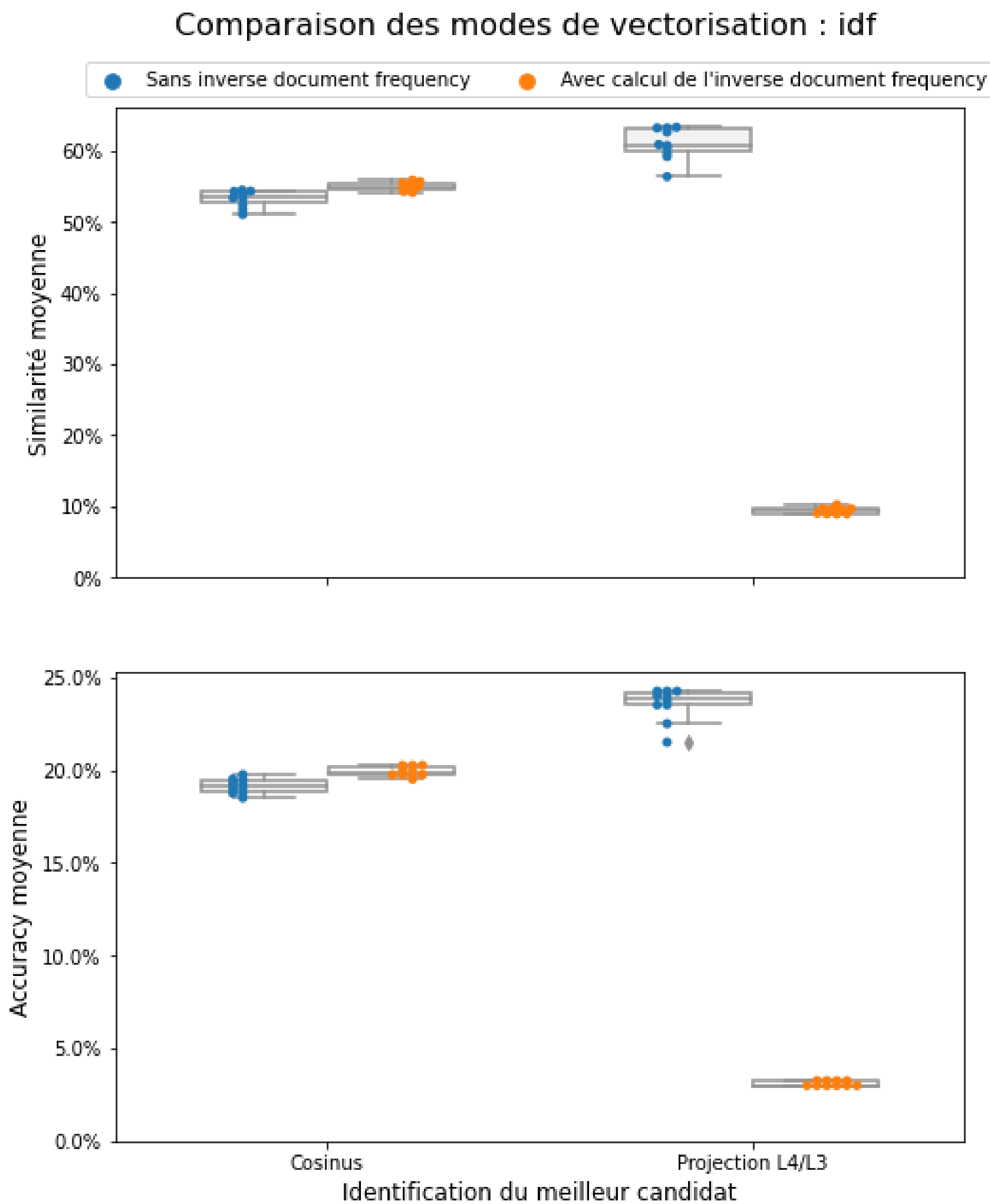
```
          bbox_transform=axs[0].transAxes,
         )

fig.suptitle("Comparaison des méthodes d'identification du meilleur candidat", fontsize=16, y=.945)

fig.savefig(Path('..') / 'img' / 'tuning_similarite.png', bbox_inches='tight')
```



[57]:
```
result_df.columns
```

[57]: Index(['Unnamed: 0', 'mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'similarity',
       'split_func', 'split0_test_similarity', 'split1_test_similarity', 'split2_test_similarity',
       'split3_test_similarity', 'split4_test_similarity', 'split5_test_similarity',
       'split6_test_similarity', 'split7_test_similarity', 'mean_test_similarity', 'std_test_similarity',
       'rank_test_similarity', 'split0_test_accuracy', 'split1_test_accuracy', 'split2_test_accuracy',
       'split3_test_accuracy', 'split4_test_accuracy', 'split5_test_accuracy', 'split6_test_accuracy',
       'split7_test_accuracy', 'mean_test_accuracy', 'std_test_accuracy', 'rank_test_accuracy', 'run',
       'projected_norm', 'source_norm', 'count_vect_type', 'scoring', 'embedding_method', 'stop_words',
       'ngram_range', 'strip_accents', 'binary', 'use_idf', 'simil_kind'],
      dtype='object')

[58]:
```python
fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[result_df['run'] == 7],
         'x': 'similarity',
         'hue': 'use_idf',
        }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
        #                   color='blue',
                          ax=axs[i],
                         )

    sns.boxplot(**parms,
                y=feature,
                ax=axs[i],
                color='white',
                width=.6,
                )

    axs[i].set_ylim(0,)
    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    axs[i].get_legend().remove()
    axs[i].set_xlabel('')

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Cosinus', 'Projection L4/L3'])

fig.legend(handles=axs[0].get_legend_handles_labels()[0][2:],
           labels=["Sans inverse document frequency", "Avec calcul de l'inverse document frequency"],
           loc='center',
           ncol=2,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
          )

fig.suptitle('Comparaison des modes de vectorisation : idf', fontsize=16, y=.95)

fig.savefig(Path('..') / 'img' / 'tuning_idf.png', bbox_inches='tight')
```

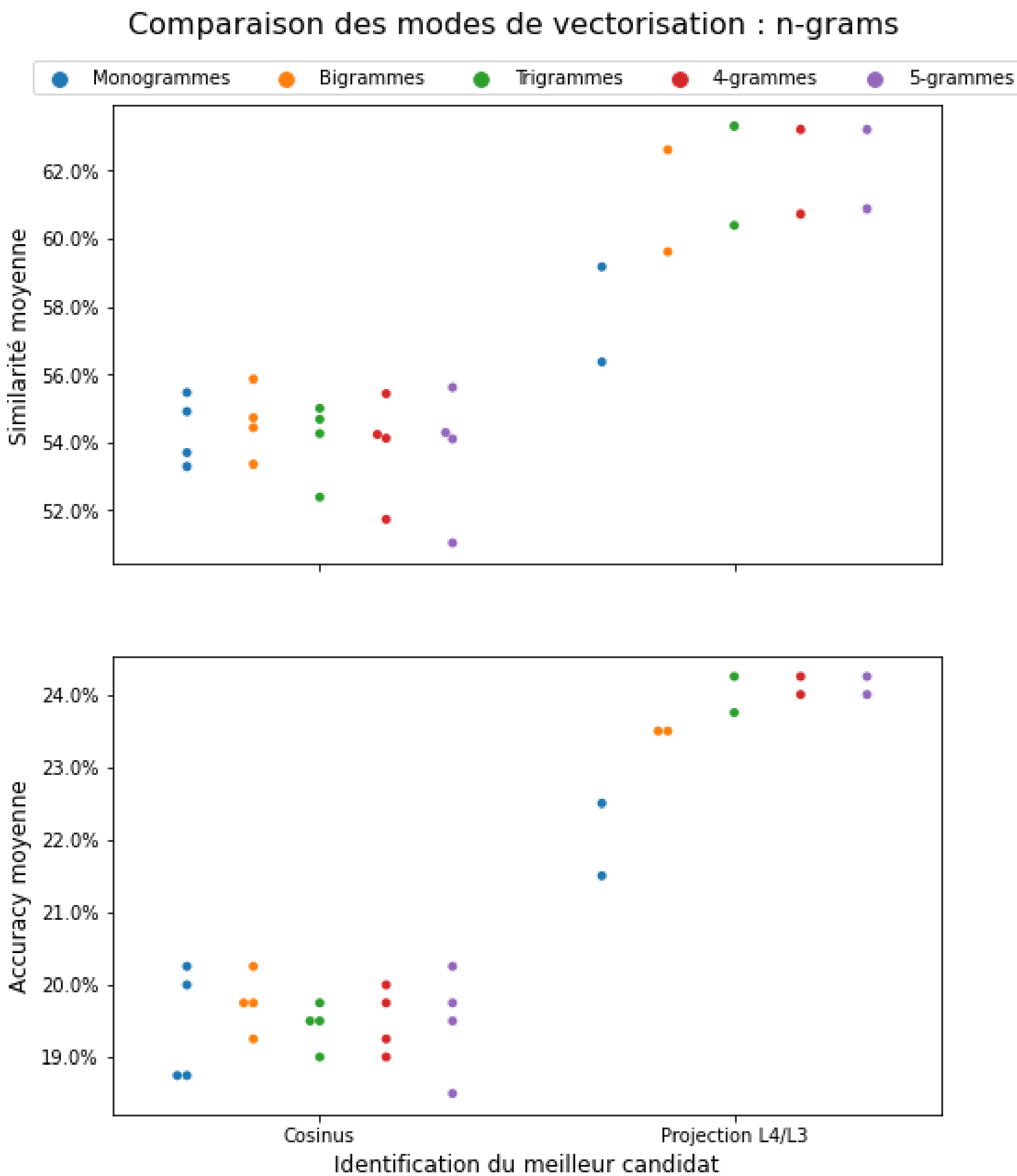


[59]:
```
result_df['use_idf']
```

[59]:
```
0      False
1      False
2      False
       …
229    False
230    False
231    False
Name: use_idf, Length: 232, dtype: bool
```

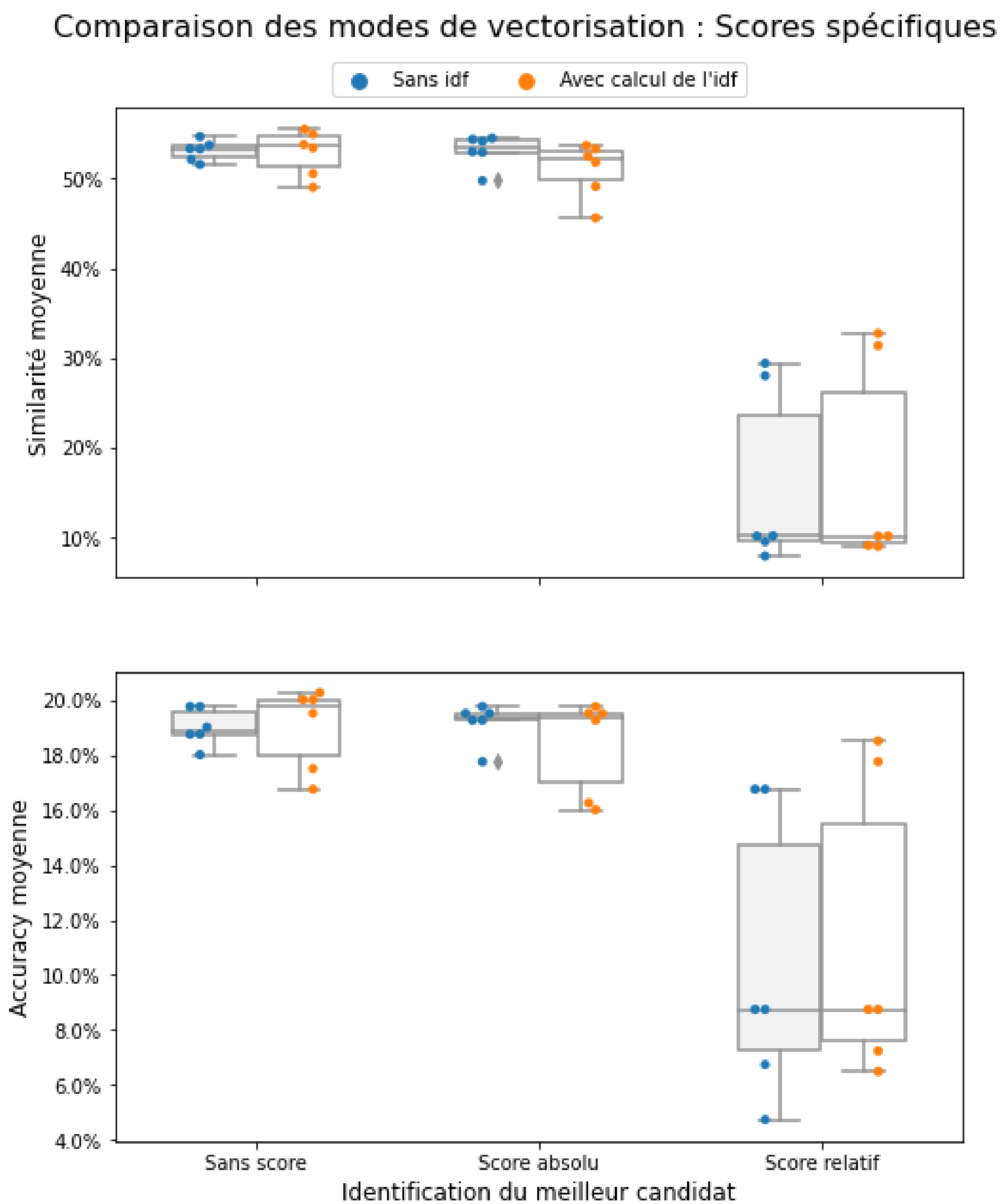
[60]:

```python
fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[(result_df['run'] == 7) &
                               ((result_df['similarity'] == 'cosine') | # & result_df['use_idf'] |
                                (result_df['similarity'] == 'projection') & ~result_df['use_idf'])
                              ],
         'x': 'similarity',
         'hue': 'ngram_range',
        }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
        #                   color='blue',
                          ax=axs[i],
                         )

#     sns.boxplot(**parms,
#                 y=feature,
#                 ax=axs[i],
#                 color='white',
#                 width=.6,
#                 )

#     axs[i].set_ylim(0,)
    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    axs[i].get_legend().remove()
    axs[i].set_xlabel('')

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Cosinus', 'Projection L4/L3'])

fig.legend(handles=axs[0].get_legend_handles_labels()[0][:],
           labels=['Monogrammes', 'Bigrammes', 'Trigrammes', '4-grammes', '5-grammes'],
           loc='center',
           ncol=5,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
          )

fig.suptitle('Comparaison des modes de vectorisation : n-grams', fontsize=16, y=.95)

fig.savefig(Path('..') / 'img' / 'tuning_ngrams.png', bbox_inches='tight')
```



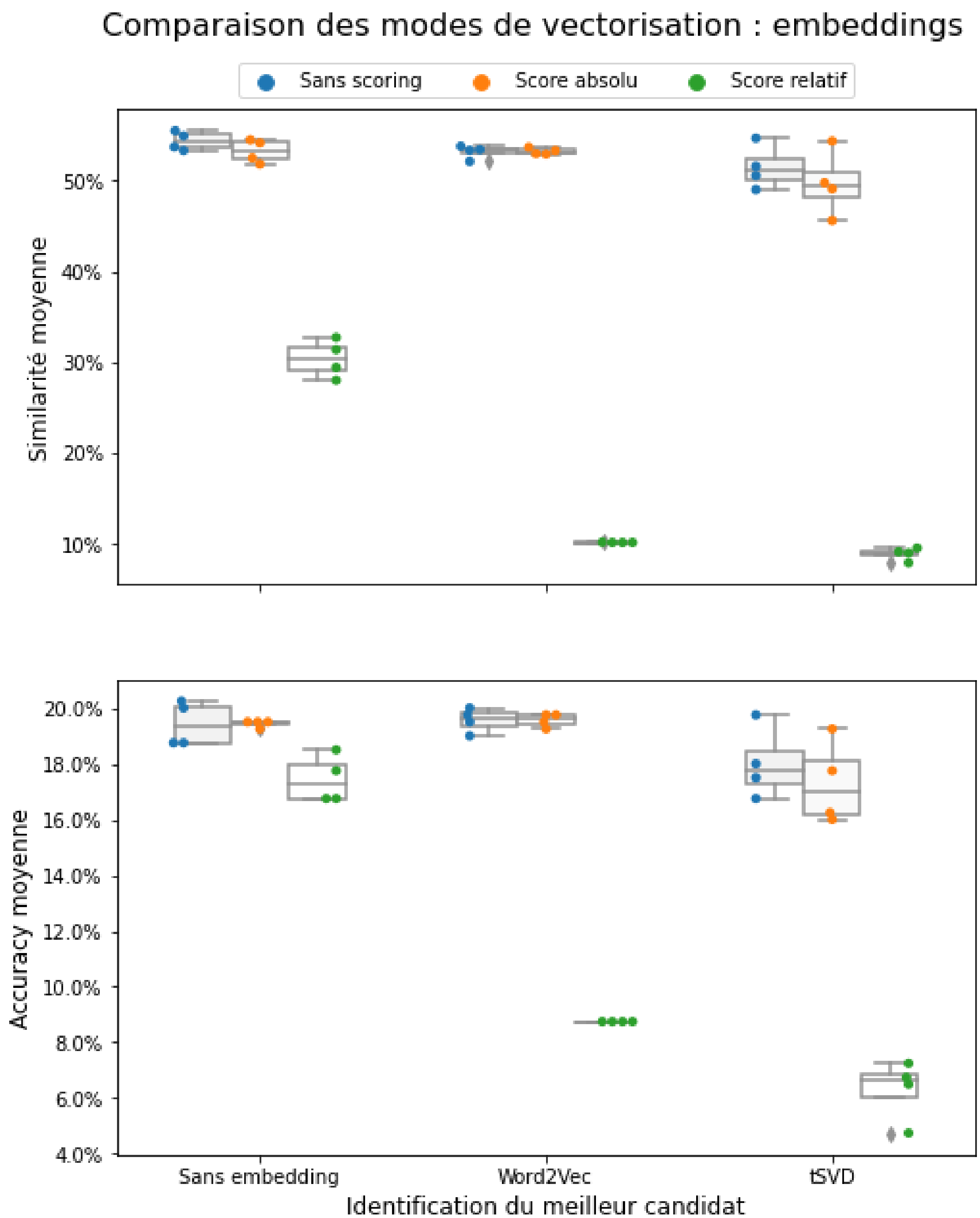
[61]:
```
fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[(result_df['run'] == 5)
                              ],
         'x': 'scoring',
         'hue': 'use_idf',
        }

for i, feature in enumerate(feats):
    swarm = sns.swarmplot(**parms,
                          y=feature,
```

```
                          dodge=True,
        #                   color='blue',
                          ax=axs[i],
                         )

    sns.boxplot(**parms,
                y=feature,
                ax=axs[i],
                color='white',
                width=.6,
                )

#     axs[i].set_ylim(0,)
    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    axs[i].get_legend().remove()
    axs[i].set_xlabel('')

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Sans score', 'Score absolu', 'Score relatif'])

fig.legend(handles=axs[0].get_legend_handles_labels()[0][2:],
           labels=["Sans idf", "Avec calcul de l'idf"],
           loc='center',
           ncol=5,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
          )

fig.suptitle('Comparaison des modes de vectorisation : Scores spécifiques', fontsize=16, y=.95)

fig.savefig(Path('..') / 'img' / 'tuning_score.png', bbox_inches='tight')
```



[62]:
```python
fig, axs = plt.subplots(nrows= 2, figsize=(8,10), sharex=True)

feats = ['mean_test_similarity', 'mean_test_accuracy']

parms = {'data': result_df.loc[(result_df['run'] == 5)
                               ],
         'x': 'embedding_method',
         'hue': 'scoring',
        }

for i, feature in enumerate(feats):
```

```python
    swarm = sns.swarmplot(**parms,
                          y=feature,
                          dodge=True,
        #                   color='blue',
                          ax=axs[i],
                         )

    sns.boxplot(**parms,
                y=feature,
                ax=axs[i],
                color='white',
                width=.6,
                )

#     axs[i].set_ylim(0,)
    axs[i].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
    axs[i].get_legend().remove()
    axs[i].set_xlabel('')

axs[1].set_xlabel('Identification du meilleur candidat', fontsize=12)
axs[0].set_ylabel('Similarité moyenne', fontsize=12)
axs[1].set_ylabel('Accuracy moyenne', fontsize=12)
axs[1].set_xticklabels(['Sans embedding', 'Word2Vec', 'tSVD'])

fig.legend(handles=axs[0].get_legend_handles_labels()[0][3:],
           labels=["Sans scoring", "Score absolu", "Score relatif"],
           loc='center',
           ncol=5,
           bbox_to_anchor=(0, 1, 1, 0.12),
           bbox_transform=axs[0].transAxes,
          )

fig.suptitle('Comparaison des modes de vectorisation : embeddings', fontsize=16, y=.95)

fig.savefig(Path('..') / 'img' / 'tuning_embedding.png', bbox_inches='tight')
```



1.3 Evaluation finale

On évalue la performance du modèle avec les meilleurs paramètres sur le set de test, après entraînement sur le set d'entrainement.

[73]:
```
parm_dict = {'Splitter__splitter_func': splitter_funcs[2],
             'SimilaritySelector__count_vect_type': 'TfidfVectorizer',
             'SimilaritySelector__similarity': 'projection',
             'SimilaritySelector__source_norm': 'l4',
```

```
            'SimilaritySelector__projected_norm': 'l3',
            'SimilaritySelector__count_vect_kwargs': {'ngram_range': (1, 3),
                                                       'stop_words': stop_words,
                                                       'strip_accents': 'unicode',
                                                       'binary': True,
                                                       'use_idf': False,
                                                      }




           }

process_pipe.set_params(**parm_dict)
process_pipe.fit(train, train['ingredients'])
print(f"Levenshtein similarity at final evaluation: {lev_scorer(process_pipe, test, test['ingredients']):.
↪2%}")
print(f"Accuracy at final evaluation: {custom_accuracy(process_pipe, test, test['ingredients']):.2%}")
```

```
Launching 8 processes.
Launching 8 processes.
Levenshtein similarity at final evaluation: 67.18%
Launching 8 processes.
Accuracy at final evaluation: 27.00%
```

[82]:
```
predicted = process_pipe.predict(test)
lev_sim = partial(text_similarity, similarity='levenshtein')
```

```
Launching 8 processes.
```

[103]:
```
comparison = (predicted.rename('Predicted')
                       .to_frame()
                       .join(test['ingredients'])
                       .rename({'ingredients': 'Target'}, axis=1))
comparison['Similarity'] = comparison.apply(lambda x: f"{lev_sim(x['Predicted'], x['Target']):.2%}", axis=1)
comparison.sample(5)
```

[103]:
```
                                                                              Predicted  \
uid
5cee689e-6fb1-493c-b232-1d8fb1f88a57  Flageolets verts. Jus : eau, sel, affermissant…
df1caa23-9714-4659-803b-33501d64eead  Liste des Ingrédients:\nsucre, pâte de cacao, …
6dfae8fd-6111-4a57-862e-c20a39a195e0  Farine de BLÉ 63%, eau, sucre, huile de colza,…
f45db604-11ad-4756-aeab-3a5a1a34f914  Ingrédients: sucre; sirop de glucose; dextrose…
6bdb201d-5879-423b-96b8-3ae88b857818  Liste d’ingrédients : vinaigre d’alcool, pimen…

                                                                                 Target Similarity
uid
5cee689e-6fb1-493c-b232-1d8fb1f88a57  Flageolets verts. Jus : eau, sel, affermissant…    100.00%
df1caa23-9714-4659-803b-33501d64eead  sucre, pâte de cacao, beurre de cacao, cacao m…     69.66%
6dfae8fd-6111-4a57-862e-c20a39a195e0  Farine de BLÉ 63%, eau, sucre, huile de colza,…     70.35%
f45db604-11ad-4756-aeab-3a5a1a34f914  sucre; sirop de glucose; dextrose; gélatine; a…     93.58%
6bdb201d-5879-423b-96b8-3ae88b857818  vinaigre d'alcool, piment jalapeno, eau, sel, …     81.25%
```

[123]:
```
with pd.option_context("max_colwidth", 2100):
    tex_str = (
    comparison.replace(r'^\s*$', np.nan, regex=True)
              .to_latex(index=False,
                        index_names=False,
                        column_format='p{9cm}p{9cm}c',
                        na_rep='<rien>',
                        longtable=True,
                        header=["Liste d'ingrédients prédite", "Liste d'ingrédients cible", "Sim."],
                        label='tbl:final_prediction',
                        caption="Prédictions du meilleur modèle sur le set de test",
                       )
              .replace(r'\textbackslash n', r' \newline ')
```

```
                .replace(r'\\', r'\\ \hline')
    )
# with open(Path('..') / 'tbls' / 'final_prediction.tex', 'w') as file:
#     file.write(tex_str)
```