

Text Mining Assignment

Elena Montenegro, Fernando Freire

May 11, 2019

Contents

1	Modules importation and data loading	2
1.1	Data split	4
2	Part I. Construction of an automatic classifier	4
2.1	Strategy: Hyper-Grid Search	5
2.2	Best classifier	8
2.3	Pipelines	8
2.3.1	Find additional stopwords	8
2.3.2	Pipelining methods	9
2.3.3	Hyper-Grid methods	13
2.4	Main process with prefixed parameters	15
2.5	Main process with grid search parameters	18
2.5.1	Example for enrichment with vectorizer grid	18
2.5.2	Example for enrichment from reducer grid	19
2.5.3	Example for enrichment for classifier grid	22
2.5.4	All in one	23
3	Part 2: Construction of a clustering of biology documents	25
3.1	Strategy (tips)	26
3.2	Best cluster	27
3.3	Main process with prefixed parameters	27
3.4	Main process with grid search parameters	28
3.5	Reference process	30

1 Modules importation and data loading

Script 1.0.1 (python)

```
1 import warnings
2 warnings.filterwarnings('ignore')
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import pandas as pd
6 import sys
7 import seaborn as sns
8 %matplotlib inline
9 from sklearn.feature_extraction.text import CountVectorizer
10 from sklearn.feature_extraction.text import TfidfTransformer
11
12 from sklearn.naive_bayes import MultinomialNB
13 from sklearn.decomposition import TruncatedSVD# SVD = Singular Value Descomposition
14 from sklearn.model_selection import GridSearchCV
15 from sklearn.feature_extraction.text import CountVectorizer
16 from sklearn.feature_extraction.text import TfidfVectorizer
17 from sklearn.preprocessing import StandardScaler, Normalizer, MinMaxScaler, MaxAbsScaler
18 from sklearn.linear_model import LogisticRegression
19 from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif
20 from sklearn.pipeline import Pipeline
21 from sklearn.model_selection import train_test_split
22 from sklearn import metrics
23 from sklearn.svm import SVC, LinearSVC
24 from sklearn.tree import DecisionTreeClassifier
25 from sklearn.neighbors import KNeighborsClassifier
26 from sklearn import tree
27 from sklearn.feature_extraction import stop_words
28 from sklearn.base import TransformerMixin
29 from sklearn.cluster import KMeans
30 from sklearn.metrics import calinski_harabaz_score, accuracy_score
31 from sklearn.preprocessing import Normalizer, LabelBinarizer, OneHotEncoder
32 from sklearn.metrics import make_scorer
33
34 random_state=0
```

Script 1.0.2 (python)

```
1 # Data loading
2 NROWS = sys.maxsize
3 #NROWS = 100
4 ## Negative dataset
5 df_neg = pd.read_csv('./practica_clase/PRECISION_MEDICINE/negative_training_abstracts.tsv',
6     ↪ sep='\t',
7     header=None, nrows = NROWS)
8 df_neg.columns = ['Accession number', 'Title', 'Abstract']
9 df_neg['Label'] = '0' #'neg'
10
```

```

11 display(df_neg.head())
12
13 corpus_neg = list(df_neg['Abstract'].values)
14 ### len(corpus_neg) # 4078
15
16 ## Positive
17 df_pos = pd.read_csv('./practica_clase/PRECISION_MEDICINE/positive_training_abstracts.tsv',
18   → sep='\t',
19   header=None, nrows = NROWS)
20
21 df_pos.columns = ['Accession number', 'Title', 'Abstract']
22 df_pos['Label'] = '1' # 'pos'
23 display(df_pos.head())
24
25 # Add corpus
26 df_corpus = df_neg.append(df_pos)
27 display(df_corpus.head())
28
29 # len(corpus) # 8156
30
31 labels = df_corpus['Label']
32 corpus = df_corpus['Abstract']
33 # len(labels) # 8156
34
35 print(len(corpus), len(labels))

```

	Accession number	Title \
0	29606186	Can reactivity and regulation in infancy predi...
1	29471205	Fabrication of bioinspired, self-cleaning supe...
2	29175165	Functional properties of chickpea protein isol...
3	29098524	Mechanical dyssynchrony alters left ventricula...
4	27507285	Reducing the width of confidence intervals for...

	Abstract	Label
0	A need to identify early infant markers of lat...	0
1	The mechanical properties, corrosion-resistanc...	0
2	In the present study, the effect of Refractanc...	0
3	The impact of left bundle branch block (LBBB) ...	0
4	In the last decade, it has been shown that an ...	0

	Accession number	Title \
0	27829177	A naturally occurring variant of HPV-16 E7 exe...
1	27806271	Functional Analysis of Orail Concatemers Suppo...
2	27796307	KAT2A/KAT2B-targeted acetylome reveals a role ...
3	27795438	The Cellular DNA Helicase ChlR1 Regulates Chro...
4	27794539	Human R1441C LRRK2 regulates the synaptic vesi...

Abstract Label

0	Human Papillomavirus E6 and E7 play critical r...	1
1	Store-operated Ca(2+) entry occurs through the...	1
2	Lysine acetylation is a widespread post-transl...	1
3	In papillomavirus infections, the viral genome...	1
4	Mutations in leucine-rich repeat kinase 2 (LRR...	1

	Accession number	Title \
0	29606186	Can reactivity and regulation in infancy predi...
1	29471205	Fabrication of bioinspired, self-cleaning supe...
2	29175165	Functional properties of chickpea protein isol...
3	29098524	Mechanical dyssynchrony alters left ventricula...
4	27507285	Reducing the width of confidence intervals for...

	Abstract Label
0	A need to identify early infant markers of lat... 0
1	The mechanical properties, corrosion-resistanc... 0
2	In the present study, the effect of Refractanc... 0
3	The impact of left bundle branch block (LBBB) ... 0
4	In the last decade, it has been shown that an ... 0

Output

8156 8156

1.1 Data split

Script 1.1.1 (python)

```
1 TEST_SIZE = 0.33
2 X_train, X_test, y_train, y_test = train_test_split(
3     corpus, labels, test_size=TEST_SIZE, random_state=random_state)
```

2 Part I. Construction of an automatic classifier

The following parameters can be adjusted in order to try to maximize the quality of the classifier:

- In function TfidfVectorizer:
 - Parameters that affect the vocabulary quality:
 - * List of stopwords (one of the options is setting it to None)
 - * maxfeatures
 - * max_df, min_df
 - Norm (none, 'l1' or 'l2')

- In Latent Semantic Analysis (LSA):
 - n_components
 - not performing LSA
- Classifier model:
 - You can use strategies included in some of the notebooks we used
 - * Logistic Regression,
 - * Naïve Bayes,
 - * decision trees,
 - * SVC
 - * or others you learnt from the Machine Learning course (k-nn, neural networks, etc.)

The goal is not to check all possible combinations of these parameters but respond to these questions:

- Which tips can you give about constructing an automatic text classifier? What do you recommend to do? What do you recommend not to do?
- What is the best classifier you have obtained?

Your responses to these questions should be illustrated with tables and/or figures and/or screen captures.

2.1 Strategy: Hyper-Grid Search

To approach this project we have decided to use the versatility of the *pipeline* objects from the *sklearn* package, surrounding it with a set of own methods to provide it with even greater dynamism.

This strategy has finally led us to develop a parameter adjustment utility that we have named it as **Hyper-Grid Search** or abbreviated **HG**. Although it is closely related to the need that emanates from the current project to explore the accuracy of a set of reducers combined with a set of classifiers, it could be generalized to any exploration scenario. We even visualize it as a very useful tool to adjust an ensemble of reducers-classifiers. Actually the full name would be Hyper Grid search by transfer of prior parameter knowledge that summarizes its ability to start a search taking reference to a set of previously adjusted parameters.

They are based on the current implementation aimed at adjusting the parameters of a Cartesian product of reducers and classifiers. These are the ones we handle but it is trivial to add new ones in the *create_text_pipeline* method

In the mentioned method its correspondence with the functions of *sklearn* can be easily visualized. We will describe the main functionalities.

Metrics dataframe The problem is to calculate the metrics for all the combinations of reducers and classifiers. In essence it is about filling this dataframe with the computed performance metrics and the parameters in each reducer-classifier combination.

reducer	classifier	precision	recall	f1-score	support
svd	knn	0.9655	0.9655	0.9655	2692
kbest	knn	0.8767	0.8767	0.8767	2692
percentile	knn	0.8544	0.8544	0.8544	2692
none	knn	0.5100	0.5100	0.5100	2692

Obtaining prior knowledge

This step is not Hyper-Grid itself in its current implementation. It consists in calculating all the metrics

for all the reducer-classifier combinations from a set of initial fixed parameters. We do this using the *process_classifications* method. By default it calculates all crosses reducers + classifiers but the lists can be filtered at the input and thus operate on more limited sets of data.

For example:

```
param_ini = {
    'vect__norm': None,
    'vect__smooth_idf': True,
    'vect__sublinear_tf': True,
    'vect__max_features': 1000,
    'vect__min_df': 1,
    'vect__max_df': 1.,
    'vect__stop_words': 'english',
    'vect__strip_accents' : 'unicode',
    'vect__analyzer' : 'word',
    'vect__ngram_range' : (1, 2),
    'scaler' : None,
    'red_kbest__k' : 5,
    'red_percentile__score_func' : f_classif,
    'red_percentile__percentile' : 10,
    'vect__norm': 'l2',
    'red_svd__n_components': 10,
    'clf_knn__n_neighbors' : 8,
}
```

```
df_metrics_fixed = process_classifications(X_train, y_train, X_test, y_test, param_ini,
                                          reducers=reducers, classifiers=classifiers)
```

About this dictionary of dictionary parameters that fits the sklearn nomenclature we will explain the nomenclature used.

1. **vect** reference to the vectorizer that is common for all reducers and classifiers.
2. **red_name** reference to a reducer of name name.
3. **clf_name** reference to a classifier with name name.

The prefix *red* is required to define a reducer and *clf* for classifiers for *HGS* to work properly.

Improvement of the metrics through grid search

At this point we start using HGS. For example to improve the metrics by exploring the best parameters of the vectorizer:

```
param_grid_vectorizer = {
    'vect__norm': ['l1', 'l2', None],
    'vect__max_features': [500, 1000],
    'vect__min_df': [1, 0.1, 0.2],
    'vect__max_df': [0.1, 0.2, 0.5, 1.],
    'vect__stop_words': [None, 'english', eng_and_custom_stopwords]
}
```

```
df_metrics_new = hyper_grid_search([param_grid_vectorizer], df_metrics_fixed, reducers=reducers, clas
```

As we see the function *hyper_grid_search* we pass the dataframe of metrics computed in the previous step. The function will automatically merge each previous set of fixed parameters from each reducer+classifier with the dict of parameter for grid search.

HGS firstly examines which of the reducer+classifier tuple is affected by the new values. If not affected the pipeline is not evaluated. Also the following rules are applied:

1. If vect is informed all reducer+classifiers are evaluated.
2. If a reducer is informed the classifiers are evaluated only for that reducer.
3. If a specific classifier has changes, only that classifier is evaluated.

HGS evaluate the best parameters using internally the grid search procedure of sklearn package.

Once improved the parameters, as a second step we compute the parameters of reducers, and finally the parameters of classifiers, to obtain a final dataframe that contains all the improved metric values and the parameters used.

Of course it's possible to run more improvements adjusting again vectorizer, reducer or classifiers.

Also it's possible to launch HGS with all grid search at once:

```
param_grid_vectorizer = {
    'vect__norm': ['l1', 'l2', None],
    'vect__max_features': [500, 1000],
    'vect__min_df': [0.0, 0.1, 0.2],
    'vect__max_df': [0.1, 0.2, 0.5, 1.],
    'vect__stop_words': [None, 'english', eng_and_custom_stopwords]
}

# Then we adjust the reducer parameters
param_grid_reducers = [
    {
        'red_svd__n_components' : [2, 3, 10, 30, 40, 100],
        'red_kbest__k' : [5, 8, 10],
        'red_percentile__score_func' : [f_classif],
        'red_percentile__percentile' : [5, 10]
    }
]

param_grid_classifiers = {
    'clf_knn__n_neighbors' : [2, 5, 8, 10, 12, 24]
}

df_metrics_all = hyper_grid_search([param_grid_vectorizer, param_grid_reducers, param_grid_classifiers],
                                   df_metrics_fixed, reducers=reducers, classifiers=classifiers)
```

Also it's possible to compute one unique grid, but the cross product with all the desired values to search of all parameters would be impractical for the processing power of the computer.

This strategy surely is not the one that would throw the best values for accuracy, because not all the parameter are computed at once, but we think it's a good trade-off between processing time and performance. This is our main recommendation.

Other good strategies that we recommend is to compute in first place reduced versions of the datasets (with less samples) and to use the parameters as a previous knowledge for the whole dataset.

2.2 Best classifier

2.3 Pipelines

2.3.1 Find additional stopwords

Script 2.3.1 (python)

```
1 def get_top_n_words(corpus, n=None):
2     """
3     List the top n words in a vocabulary according to occurrence in a text corpus.
4     """
5     vec = CountVectorizer().fit(corpus)
6     bag_of_words = vec.transform(corpus)
7     sum_words = bag_of_words.sum(axis=0)
8     words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
9     words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
10
11     return words_freq[:n]
12
13 def improve_stop_words(X_train, n=50):
14     """
15     """
16     common_words = [i[0] for i in get_top_n_words(X_train, n)]
17     eng_and_custom_stopwords = set(list(stop_words.ENGLISH_STOP_WORDS) + common_words)
18     print("Stop words count:", len(eng_and_custom_stopwords))
19     return eng_and_custom_stopwords
```

2.3.2 Pipelining methods

Script 2.3.2 (python)

```
1 CLASSIFIERS = ['knn', 'dtree', 'nb', 'lr', 'svc', 'lsvc']
2 CLASSIFIERS_UNSUPERVISED = ['kmeans']
3 REDUCERS = ['svd', 'kbest', 'percentile', 'none']
4 CV = 4
5 VERBOSE = False
6
7 def create_text_pipeline(reducer='svd', classifier="nb"):
8     """ Create text vectorization pipeline with optional dimensionality reduction"""
9     assert reducer in REDUCERS, "ERROR: Reducer %s not supported, only %s" % (reducer,
10     ↪ REDUCERS)
11     assert classifier in CLASSIFIERS + CLASSIFIERS_UNSUPERVISED, \
12         "ERROR: Classifier %s not supported, only %s" % (classifier, CLASSIFIERS +
13         ↪ CLASSIFIERS_UNSUPERVISED)
14     pipeline = [
15         ('vect', TfidfVectorizer()),
16         ('scaler', StandardScaler())
17     ]
18     # Reduce dimensions
19     if reducer == 'svd':
```



```

18     pipeline.append(('red_svd', TruncatedSVD()))
19 elif reducer == 'kbest':
20     pipeline.append(('red_kbest', SelectKBest()))
21 elif reducer == 'percentile':
22     pipeline.append(('red_percentile', SelectPercentile()))
23 elif reducer == 'none':
24     pass
25
26 # Classify
27 if classifier == "nb":
28     if reducer == 'svd':
29         pipeline.append(('clf_nb_scaler', MinMaxScaler()))
30     elif reducer == 'kbest':
31         pipeline.append(('clf_nb_scaler', MaxAbsScaler()))
32     elif reducer == 'percentile':
33         pipeline.append(('clf_nb_scaler', MaxAbsScaler()))
34     elif reducer == 'none':
35         pass
36     pipeline.append(('clf_' + classifier, MultinomialNB()))
37 elif classifier == "lr":
38     pipeline.append(('clf_' + classifier, LogisticRegression()))
39 elif classifier == "svc":
40     pipeline.append(('clf_' + classifier, SVC()))
41 elif classifier == "lsvc":
42     pipeline.append(('clf_' + classifier, LinearSVC()))
43 elif classifier == "dtree":
44     pipeline.append(('clf_' + classifier, DecisionTreeClassifier()))
45 elif classifier == "knn":
46     pipeline.append(('clf_' + classifier, KNeighborsClassifier()))
47 elif classifier == "kmeans":
48     pipeline.append(('clf_kmeans_norm', Normalizer()))
49     pipeline.append(('clf_kmeans', KMeans()))
50 elif classifier == 'none':
51     pass
52
53 return Pipeline(pipeline)
54
55 def get_prediction_from_cluster(X, pipeline):
56     """ Transform cluster assignment in y_pred object """
57     def swap_label(label):
58         if label == 1:
59             return '0'
60         elif label == 0:
61             return '1'
62         else:
63             return str(label)
64     labels = pipeline.predict(X_test)
65     labels_predicted = [str(label) for label in labels]
66     predicted = pd.Series(labels_predicted)
67     accuracy = metrics.accuracy_score(y_test, predicted)
68     labels_predicted_reverse = [swap_label(label) for label in labels]
69     predicted_reverse = pd.Series(labels_predicted_reverse)

```

```

70     accuracy_reverse = metrics.accuracy_score(y_test, predicted_reverse)
71     if accuracy_reverse > accuracy: predicted = predicted_reverse
72     return predicted
73
74 def get_filtered_params(parameters, pipeline):
75     """ Filter the params that aren't related to steps in the pipeline """
76     filtered_params = {}
77     for param_key in parameters.keys():
78         if param_key.split('__')[0] in pipeline.named_steps.keys():
79             filtered_params[param_key] = parameters[param_key]
80     return filtered_params
81
82 def params2search(parameters_search, parameters_prev):
83     """ Convert params to search params """
84     # Generalize params to list of params
85     if type(parameters_search) == dict:
86         parameters = [parameters_search]
87     else:
88         parameters = parameters_search
89     search_params_set = []
90     for param_set in parameters:
91         search_params = param_set.copy()
92         for param_key in parameters_prev.keys():
93             if param_key not in param_set:
94                 search_params[param_key] = [parameters_prev[param_key]]
95         search_params_set.append(search_params)
96     return search_params_set
97
98 def get_filtered_set(parameters, pipeline):
99     """ Filter the params that aren't related to steps in the pipeline """
100     if type(parameters) == dict:
101         return get_filtered_params(parameters, pipeline)
102     else:
103         filtered_set = []
104         for param_set in parameters:
105             filtered_set.append(get_filtered_params(param_set, pipeline))
106     return filtered_set
107
108 def prediction_metrics(X_train, y_train, X_test, y_test, parameters, results, reducer="svd",
109     ↪ classifier="nb"):
110     """
111     Get performance metrics from sklearn classification report 'micro avg'
112     """
113     print("### Reducer: %s Classifier: %s" %(reducer, classifier))
114     pipeline = create_text_pipeline(reducer=reducer, classifier=classifier)
115     pipeline.set_params(**get_filtered_params(parameters, pipeline))
116     if VERBOSE: print("Pipeline", pipeline.named_steps)
117     pipeline.fit(X_train, y_train)
118     if classifier in CLASSIFIERS_UNSUPERVISED:
119         predicted = get_prediction_from_cluster(X_test, pipeline)
120     else:
121         predicted = pipeline.predict(X_test)

```

```

121 accuracy = metrics.accuracy_score(y_test, predicted)
122 print("Accuracy", accuracy)
123 clf_rep = metrics.classification_report(y_test, predicted, output_dict=True, digits=2)
124 if VERBOSE: print(clf_rep['micro avg'])
125 if VERBOSE: print(metrics.confusion_matrix(y_test, predicted))
126 print()
127
128 results.append([reducer, classifier] + \
129                 list(clf_rep['micro avg'].values()) + [parameters])
130
131 def process_classifications(X_train, y_train, X_test, y_test, parameters,
132                             classifiers=CLASSIFIERS, reducers=REDUCERS):
133     """
134     """
135     results = []
136     for classifier in classifiers:
137         for reducer in reducers:
138             prediction_metrics(X_train, y_train, X_test, y_test, parameters, results,
139                               ⇨ reducer, classifier)
140             # Group all results into a dataframe
141             df = pd.DataFrame(results, columns=['reducer', 'classifier', 'precision', 'recall',
142                                               ⇨ 'f1-score', 'support', 'params'])
143             df['classifier'].fillna('None', inplace=True)
144
145             return df
146
147 def prediction_metrics_grid(X_train, y_train, X_test, y_test, parameters_grid, results=[],
148                             reducer="svd", classifier="nb", cv=CV):
149     """
150     """
151     print("### Reducer: %s Classifier: %s" %(reducer, classifier))
152     pipeline = create_text_pipeline(reducer=reducer, classifier=classifier)
153     filtered_params = get_filtered_set(parameters_grid, pipeline)
154     #scoring = {'accuracy': make_scorer(accuracy_score), 'calinski':
155     ⇨ make_scorer(calinski_harabaz_score)}
156     scoring = {'accuracy': make_scorer(accuracy_score)}
157     grid_model = GridSearchCV(pipeline, filtered_params, cv=cv, iid=False, error_score=0,
158                               scoring=None, refit=False)
159     grid_model.fit(X_train, y_train)
160     print()
161     print("Best parameters")
162     for param_name in sorted(grid_model.best_params_.keys()):
163         print("\t%s: %r" % (param_name, grid_model.best_params_[param_name]))
164     pipeline.set_params(**grid_model.best_params_)
165     pipeline.fit(X_train, y_train)
166     if classifier in CLASSIFIERS_UNSUPERVISED:
167         predicted = get_prediction_from_cluster(X_test, pipeline)
168     else:
169         predicted = pipeline.predict(X_test)
170     print()
171     accuracy = metrics.accuracy_score(y_test, predicted)
172     print("Accuracy", accuracy)

```

```

170 clf_rep = metrics.classification_report(y_test, predicted, output_dict=True, digits=2)
171 if VERBOSE: print(clf_rep['micro avg'])
172 if VERBOSE: print(metrics.confusion_matrix(y_test, predicted))
173 print()
174 results.append([reducer, classifier] + \
175                 list(clf_rep['micro avg'].values()) + [grid_model.best_params_])
176
177
178 def process_classifications_grid(X_train, y_train, X_test, y_test, parameters, cv=CV,
179                                classifiers=CLASSIFIERS, reducers=REDUCERS):
180     """
181     """
182     results = []
183     for classifier in classifiers:
184         for reducer in reducers:
185             prediction_metrics_grid(X_train, y_train, X_test, y_test, parameters,
186                                   results, reducer, classifier, cv=cv)
187
188     # Group all results into a dataframe
189     df = pd.DataFrame(results, columns=['reducer', 'classifier', 'precision', 'recall',
190                                       'f1-score', 'support', 'params'])
191     df['classifier'].fillna('None', inplace=True)
192
193     return df

```

2.3.3 Hyper-Grid methods

Script 2.3.3 (python)

```

1 def params2search(parameters_search, parameters_prev):
2     """ Convert params to search params """
3     # Generalize params to list of params
4     if type(parameters_search) == dict:
5         parameters = [parameters_search]
6     else:
7         parameters = parameters_search
8     search_params_set = []
9     for param_set in parameters:
10         #print(param_set['vect_min_df'])
11         search_params = param_set.copy()
12         for param_key in parameters_prev.keys():
13             if param_key not in param_set.keys():
14                 #print("Key:", param_key)
15                 search_params[param_key] = [parameters_prev[param_key]]
16         search_params_set.append(search_params)
17     return search_params_set
18
19 def hyper_grid_search(grid_parameters, df_metrics_old, reducers, classifiers):
20     """
21     Main method for search
22     """

```

```

23 for step, grid_parameters in enumerate(grid_parameters):
24     #df_metrics_new = df_metrics_old[~(df_metrics_old['reducer'].isin(reducers)) &
    ↪ ~ (df_metrics_old['classifier'].isin(classifiers))]
25     df_metrics_new = pd.DataFrame()
26     df_metrics_old_cp = df_metrics_old.copy()
27     reducers_affected, classifiers_affected = params2affected(grid_parameters)
28     for reducer in reducers:
29         for classifier in classifiers:
30             print("Step", step, "Reducer", reducer, "Classifier", classifier)
31             if reducer in reducers_affected and classifier in classifiers_affected:
32                 params = list(df_metrics_old[(df_metrics_old['reducer'] == reducer)\
33                                     ↪ (df_metrics_old['classifier'] ==
    ↪ classifier)]['params'])[0]
34                 new_search_params = params2search(grid_parameters, params)
35                 #print("New parameters",new_search_params)
36                 df_metrics_tmp = process_classifications_grid(X_train, y_train, X_test,
    ↪ y_test, new_search_params,
37                                     reducers=[reducer], classifiers=[classifier])
38                 df_metrics_new = df_metrics_new.append(df_metrics_tmp)
39             else:
40                 print('not affected')
41                 df_metrics_new = df_metrics_new.append(df_metrics_old_cp[(df_metrics_old
    ↪ _cp['reducer'] ==
    ↪ reducer)\
42                                     ↪ (df_metrics_old_cp['classifier'] ==
    ↪ classifier)])
43                 df_metrics_new = df_metrics_new.append(df_metrics_old_cp[~(df_metrics_old_cp['reduce
    ↪ r'].isin(reducers) &
    ↪ df_metrics_old_cp['classifier'].isin(classifiers))])
44                 df_metrics_old = df_metrics_new
45
46     return df_metrics_new
47
48 def params2affected(parameters_search):
49     """ Decide if affected """
50     # Generalize params to list of params
51     if type(parameters_search) == dict:
52         parameters = [parameters_search]
53     else:
54         parameters = parameters_search
55     reducers = []
56     classifiers = []
57     all_reducers = False
58     all_classifiers = False
59     for param_set in parameters:
60         for param_key in param_set.keys():
61             key = param_key.split('__')[0]
62             t = key.split('_')[0]
63             if t == 'red':
64                 reducers.append(key.split('_')[1])
65                 all_classifiers = True
66             elif t == 'clf':

```

```

67         classifiers.append(key.split('_')[1])
68         all_reducers = True
69         elif t == 'vect': #all reducers and classifiers affected
70             all_classifiers = True
71             all_reducers = True
72     if all_reducers: reducers = REDUCERS
73     if all_classifiers: classifiers = CLASSIFIERS + CLASSIFIERS_UNSUPERVISED
74     return reducers, classifiers
75
76 def plot_heatmap(df,cols, figsize=(7,5)):
77     '''
78     Function for plotting cross results. It returns a heatmap
79     with the accuracy for each combination of reducers and classifiers
80     '''
81     # Selecting the data
82     DATA= np.split(df[cols[2]].values,
83                     df[cols[1]].unique().shape[0])
84
85     df_ =pd.DataFrame(data=DATA, index=list(df[cols[1]].unique()),
86                       columns=list(df[cols[0]].unique()))
87
88     sns.set(rc={'figure.figsize':figsize})
89     sns.heatmap(df_.T, cmap="Blues", annot=True, cbar=True,
90                cbar_kws={'label': 'Accuracy'}, fmt="0.3f")
91
92     plt.xticks(np.arange(df_.index.shape[0])+0.5,
93               horizontalalignment='center', size=13)
94
95     plt.yticks(np.arange(df_.columns.shape[0])+0.2,
96               ha='center', size=13)
97
98     plt.title('Accuracy for each reducer and classifier combinations', size=14)
99     plt.xlabel(cols[1], size=13)
100    plt.ylabel(cols[0], size=13)
101    plt.show()

```

2.4 Main process with prefixed parameters

Script 2.4.1 (python)

```

1  VERBOSE = False
2  # More stop words
3  eng_and_custom_stopwords = improve_stop_words(X_train, 200)
4  reducers=REDUCERS
5  classifiers = CLASSIFIERS
6
7  # First set of parameters
8  param_ini = {
9      'vect__norm': None,
10     'vect__smooth_idf': True,

```

```

11     'vect__sublinear_tf': True,
12     'vect__max_features': 1000,
13     'vect__min_df': 1,
14     'vect__max_df': 1.,
15     'vect__stop_words': 'english',
16     'vect__strip_accents' : 'unicode',
17     'vect__analyzer' : 'word',
18     #'vect__token_pattern': r'\w{1,}',
19     'vect__ngram_range' : (1, 2),
20     'scaler' : None,
21     'red_kbest__k' : 5,
22     'red_percentile__score_func' : f_classif,
23     'red_percentile__percentile' : 10,
24     'vect__norm': 'l2',
25     'red_svd__n_components': 10,
26     'clf_knn__n_neighbors' : 8,
27 }
28
29 df_metrics_fixed = process_classifications(X_train, y_train, X_test, y_test, param_ini,\
30                                           reducers=reducers, classifiers=classifiers)

```

Output

```

Stop words count: 449
### Reducer: svd    Classifier: knn
Accuracy 0.9654531946508172

### Reducer: kbest   Classifier: knn
Accuracy 0.8766716196136701

### Reducer: percentile  Classifier: knn
Accuracy 0.8543833580980683

### Reducer: none    Classifier: knn
Accuracy 0.5100297176820208

### Reducer: svd    Classifier: dtree
Accuracy 0.9483655274888558

### Reducer: kbest   Classifier: dtree
Accuracy 0.861812778603269

### Reducer: percentile  Classifier: dtree
Accuracy 0.9201337295690936

### Reducer: none    Classifier: dtree
Accuracy 0.9182763744427934

### Reducer: svd    Classifier: nb
Accuracy 0.9647102526002972

```

```

### Reducer: kbest   Classifier: nb
Accuracy 0.34583952451708766

### Reducer: percentile   Classifier: nb
Accuracy 0.9309063893016345

### Reducer: none   Classifier: nb
Accuracy 0.9565378900445766

### Reducer: svd   Classifier: lr
Accuracy 0.9702823179791976

### Reducer: kbest   Classifier: lr
Accuracy 0.8610698365527489

### Reducer: percentile   Classifier: lr
Accuracy 0.9598811292719168

### Reducer: none   Classifier: lr
Accuracy 0.975111441307578

### Reducer: svd   Classifier: svc
Accuracy 0.9691679049034175

### Reducer: kbest   Classifier: svc
Accuracy 0.8610698365527489

### Reducer: percentile   Classifier: svc
Accuracy 0.9528231797919762

### Reducer: none   Classifier: svc
Accuracy 0.49925705794947994

### Reducer: svd   Classifier: lsvc
Accuracy 0.9706537890044576

### Reducer: kbest   Classifier: lsvc
Accuracy 0.8681277860326895

### Reducer: percentile   Classifier: lsvc
Accuracy 0.9591381872213968

### Reducer: none   Classifier: lsvc
Accuracy 0.9706537890044576

```

Script 2.4.2 (python)

```

1 pd.set_option('display.max_colwidth', 20)
2 display(df_metrics_fixed.iloc[:, :-1])

```

reducer classifier precision recall f1-score support

0	svd	knn	0.965453	0.965453	0.965453	2692
1	kbest	knn	0.876672	0.876672	0.876672	2692
2	percentile	knn	0.854383	0.854383	0.854383	2692
3	none	knn	0.510030	0.510030	0.510030	2692
4	svd	dtree	0.948366	0.948366	0.948366	2692
5	kbest	dtree	0.861813	0.861813	0.861813	2692
6	percentile	dtree	0.920134	0.920134	0.920134	2692
7	none	dtree	0.918276	0.918276	0.918276	2692
8	svd	nb	0.964710	0.964710	0.964710	2692
9	kbest	nb	0.345840	0.345840	0.345840	2692
10	percentile	nb	0.930906	0.930906	0.930906	2692
11	none	nb	0.956538	0.956538	0.956538	2692
12	svd	lr	0.970282	0.970282	0.970282	2692
13	kbest	lr	0.861070	0.861070	0.861070	2692
14	percentile	lr	0.959881	0.959881	0.959881	2692
15	none	lr	0.975111	0.975111	0.975111	2692
16	svd	svc	0.969168	0.969168	0.969168	2692
17	kbest	svc	0.861070	0.861070	0.861070	2692
18	percentile	svc	0.952823	0.952823	0.952823	2692
19	none	svc	0.499257	0.499257	0.499257	2692
20	svd	lsvc	0.970654	0.970654	0.970654	2692
21	kbest	lsvc	0.868128	0.868128	0.868128	2692
22	percentile	lsvc	0.959138	0.959138	0.959138	2692
23	none	lsvc	0.970654	0.970654	0.970654	2692

2.5 Main process with grid search parameters

2.5.1 Example for enrichment with vectorizer grid

Script 2.5.1 (python)

```

1 # First we adjust the vectorizer parameters
2 param_grid_vectorizer = {
3     'vect__norm': ['l1', 'l2', None],
4     'vect__max_features': [500, 1000],
5     'vect__min_df': [1, 0.1, 0.2],
6     'vect__max_df': [0.1, 0.2, 0.5, 1.],
7     'vect__stop_words': [None, 'english', eng_and_custom_stopwords]
8 }
9 classifiers=['lsvc','knn', 'lr']
10 df_metrics_new = hyper_grid_search([param_grid_vectorizer], df_metrics_fixed,
    ↪ reducers=reducers, classifiers=classifiers)

```

Script 2.5.2 (python)

```

1 display(df_metrics_new.iloc[:,-1])

```

reducer classifier precision recall f1-score support

0	svd	lsvc	0.969697	0.969697	0.969697	33
0	svd	knn	0.969697	0.969697	0.969697	33
0	svd	lr	0.969697	0.969697	0.969697	33
0	kbest	lsvc	0.818182	0.818182	0.818182	33
0	kbest	knn	0.939394	0.939394	0.939394	33
0	kbest	lr	0.848485	0.848485	0.848485	33
2	svd	dtree	0.939394	0.939394	0.939394	33
3	kbest	dtree	0.878788	0.878788	0.878788	33
4	svd	nb	0.909091	0.909091	0.909091	33
5	kbest	nb	0.575758	0.575758	0.575758	33
8	svd	svc	0.454545	0.454545	0.454545	33
9	kbest	svc	0.454545	0.454545	0.454545	33

2.5.2 Example for enrichment from reducer grid

Script 2.5.3 (python)

```

1 param_grid_reducers = [
2     {
3         'red_svd__n_components' : [2, 3, 10, 30, 40, 100],
4         'red_kbest__k' : [1, 2, 3, 5, 10, 20, 50, 60],
5         'red_percentile__score_func' : [f_classif],
6         'red_percentile__percentile' : [5, 10]
7     }
8 ]
9
10 df_metrics_reducers = hyper_grid_search([param_grid_reducers], df_metrics_new,
    ↪ reducers=reducers, classifiers=classifiers)

```

Output

```

Step 0 Reducer svd Classifier lsvc
### Reducer: svd Classifier: lsvc

Best parameters
  red_svd__n_components: 10
  scaler: None
  vect__analyzer: 'word'
  vect__max_df: 1.0
  vect__max_features: 500
  vect__min_df: 0.1
  vect__ngram_range: (1, 2)
  vect__norm: 'l2'
  vect__smooth_idf: True
  vect__stop_words: None
  vect__strip_accents: 'unicode'
  vect__sublinear_tf: True

Accuracy 0.9696969696969697

```

Step 0 Reducer svd Classifier knn
Reducer: svd Classifier: knn

Best parameters

clf_knn__n_neighbors: 8
red_svd__n_components: 3
scaler: None
vect__analyzer: 'word'
vect__max_df: 1.0
vect__max_features: 500
vect__min_df: 0.1
vect__ngram_range: (1, 2)
vect__norm: 'l2'
vect__smooth_idf: True
vect__stop_words: None
vect__strip_accents: 'unicode'
vect__sublinear_tf: True

Accuracy 0.9090909090909091

Step 0 Reducer svd Classifier lr
Reducer: svd Classifier: lr

Best parameters

red_svd__n_components: 10
scaler: None
vect__analyzer: 'word'
vect__max_df: 1.0
vect__max_features: 500
vect__min_df: 0.1
vect__ngram_range: (1, 2)
vect__norm: 'l2'
vect__smooth_idf: True
vect__stop_words: None
vect__strip_accents: 'unicode'
vect__sublinear_tf: True

Accuracy 0.9696969696969697

Step 0 Reducer kbest Classifier lsvc
Reducer: kbest Classifier: lsvc

Best parameters

red_kbest__k: 20
scaler: None
vect__analyzer: 'word'
vect__max_df: 0.5
vect__max_features: 500
vect__min_df: 1
vect__ngram_range: (1, 2)
vect__norm: 'l2'

```
vect__smooth_idf: True
vect__stop_words: None
vect__strip_accents: 'unicode'
vect__sublinear_tf: True
```

Accuracy 0.8787878787878788

Step 0 Reducer kbest Classifier knn
Reducer: kbest Classifier: knn

Best parameters

```
clf_knn__n_neighbors: 8
red_kbest__k: 3
scaler: None
vect__analyzer: 'word'
vect__max_df: 0.5
vect__max_features: 500
vect__min_df: 1
vect__ngram_range: (1, 2)
vect__norm: 'l2'
vect__smooth_idf: True
vect__stop_words: 'english'
vect__strip_accents: 'unicode'
vect__sublinear_tf: True
```

Accuracy 0.9393939393939394

Step 0 Reducer kbest Classifier lr
Reducer: kbest Classifier: lr

Best parameters

```
red_kbest__k: 3
scaler: None
vect__analyzer: 'word'
vect__max_df: 0.5
vect__max_features: 500
vect__min_df: 0.2
vect__ngram_range: (1, 2)
vect__norm: 'l2'
vect__smooth_idf: True
vect__stop_words: 'english'
vect__strip_accents: 'unicode'
vect__sublinear_tf: True
```

Accuracy 0.8484848484848485

Script 2.5.4 (python)

```
1 df_metrics_reducers.iloc[:, :-1]
```

Display output

	reducer	classifier	precision	recall	f1-score	support
0	svd	lsvc	0.969697	0.969697	0.969697	33
0	svd	knn	0.909091	0.909091	0.909091	33
0	svd	lr	0.969697	0.969697	0.969697	33
0	kbest	lsvc	0.878788	0.878788	0.878788	33
0	kbest	knn	0.939394	0.939394	0.939394	33
0	kbest	lr	0.848485	0.848485	0.848485	33
2	svd	dtree	0.939394	0.939394	0.939394	33
3	kbest	dtree	0.878788	0.878788	0.878788	33
4	svd	nb	0.909091	0.909091	0.909091	33
5	kbest	nb	0.575758	0.575758	0.575758	33
8	svd	svc	0.454545	0.454545	0.454545	33
9	kbest	svc	0.454545	0.454545	0.454545	33

2.5.3 Example for enrichment for classifier grid

Script 2.5.5 (python)

```
1 parameters_classifiers = {
2     'clf_knn__n_neighbors' : [2, 5, 8, 10, 12, 20]
3 }
4
5 df_metrics_classif = hyper_grid_search([parameters_classifiers], df_metrics_reducers,\
6     reducers=reducers, classifiers=classifiers)
```

Output

```
Step 0 Reducer svd Classifier lsvc
not affected
Step 0 Reducer svd Classifier knn
### Reducer: svd Classifier: knn

Best parameters
  clf_knn__n_neighbors: 10
  red_svd__n_components: 10
  scaler: None
  vect__analyzer: 'word'
  vect__max_df: 1.0
  vect__max_features: 500
  vect__min_df: 0.1
  vect__ngram_range: (1, 2)
  vect__norm: 'l2'
  vect__smooth_idf: True
  vect__stop_words: None
  vect__strip_accents: 'unicode'
  vect__sublinear_tf: True

Accuracy 0.9696969696969697
```

Script 2.5.6 (python)

```
1 display(df_metrics_classif.iloc[:, :-1])
```

	reducer	classifier	precision	recall	f1-score	support
0	svd	lsvc	0.969697	0.969697	0.969697	33
0	svd	knn	0.969697	0.969697	0.969697	33
1	svd	dtree	0.909091	0.909091	0.909091	33
2	svd	nb	0.909091	0.909091	0.909091	33
3	svd	lr	0.939394	0.939394	0.939394	33
4	svd	svc	0.454545	0.454545	0.454545	33

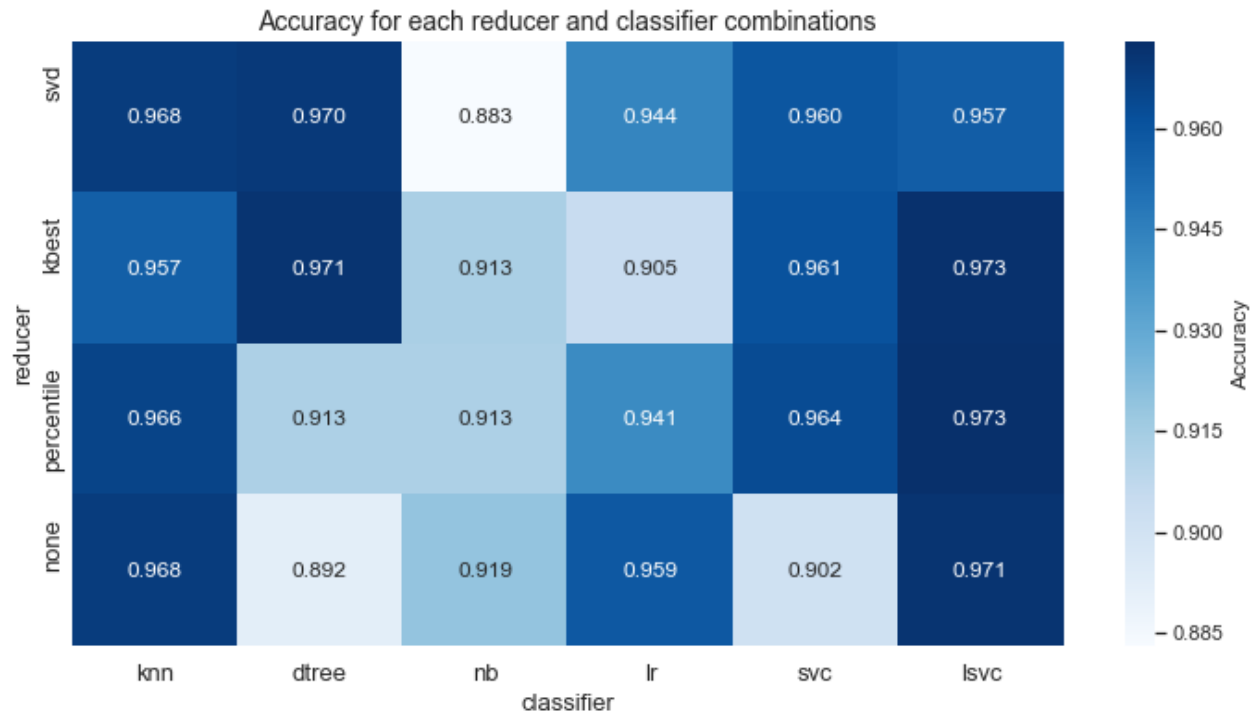
2.5.4 All in one

Script 2.5.7 (python)

```
1 # First we adjust the vectorizer parameters
2 param_grid_vectorizer = {
3     'vect__norm': ['l1', 'l2', None],
4     'vect__max_features': [500, 1000],
5     'vect__min_df': [0.0, 0.1, 0.2],
6     'vect__max_df': [0.1, 0.2, 0.5, 1.],
7     'vect__stop_words': [None, 'english', eng_and_custom_stopwords]
8 }
9
10 # Then we adjust the reducer parameters
11 param_grid_reducers = [
12     {
13         'red_svd__n_components' : [2, 3, 10, 30, 40, 100],
14         'red_kbest__k' : [5, 8, 10],
15         'red_percentile__score_func' : [f_classif],
16         'red_percentile__percentile' : [5, 10]
17     }
18 ]
19
20 param_grid_classifiers = {
21     'clf_knn__n_neighbors' : [2, 5, 8, 10, 12, 24]
22 }
23
24 df_metrics_all = hyper_grid_search([param_grid_vectorizer, param_grid_reducers,
25     ↪ param_grid_classifiers],
26                                   df_metrics_fixed, reducers=reducers,
27                                   ↪ classifiers=classifiers)
```

Script 2.5.8 (python)

```
1 plot_heatmap(df_metrics_all, cols=['reducer', 'classifier', 'precision'], figsize=(12,6))
2 display(df_metrics_all.iloc[:, :-1])
```



	reducer	classifier	precision	recall	f1-score	support
0	svd	knn	0.968425	0.968425	0.968425	2692
0	svd	dtree	0.956538	0.956538	0.956538	2692
0	svd	nb	0.966196	0.966196	0.966196	2692
0	svd	lr	0.968425	0.968425	0.968425	2692
0	svd	svc	0.969539	0.969539	0.969539	2692
0	svd	lsvc	0.971397	0.971397	0.971397	2692
0	kbest	knn	0.912704	0.912704	0.912704	2692
0	kbest	dtree	0.892273	0.892273	0.892273	2692
0	kbest	nb	0.882987	0.882987	0.882987	2692
0	kbest	lr	0.913447	0.913447	0.913447	2692
0	kbest	svc	0.912704	0.912704	0.912704	2692
0	kbest	lsvc	0.919019	0.919019	0.919019	2692
0	percentile	knn	0.943536	0.943536	0.943536	2692
0	percentile	dtree	0.904532	0.904532	0.904532	2692
0	percentile	nb	0.941308	0.941308	0.941308	2692
0	percentile	lr	0.959138	0.959138	0.959138	2692
0	percentile	svc	0.959881	0.959881	0.959881	2692
0	percentile	lsvc	0.960996	0.960996	0.960996	2692
0	none	knn	0.963596	0.963596	0.963596	2692
0	none	dtree	0.901560	0.901560	0.901560	2692
0	none	nb	0.957281	0.957281	0.957281	2692
0	none	lr	0.972511	0.972511	0.972511	2692
0	none	svc	0.972883	0.972883	0.972883	2692

0	none	lsvc	0.971397	0.971397	0.971397	2692
---	------	------	----------	----------	----------	------

3 Part 2: Construction of a clustering of biology documents

We already know the class information in our dataset (positive and negative) but we will test if an automatic clustering system discovers automatically these classes (“labels”). The objective is to learn strategies that will be very useful when we have to cluster unlabeled documents. Therefore, we “hide” this information (the real class) to the clustering algorithm.

The objective in this section is to check what are the parameters that maximize clustering’s quality. The parameters to be taken into account are:

- In function `TfidfVectorizer`:
 - Vocabulary (larger or smaller)
 - Norm (none, ‘l1’ or ‘l2’)
- In Latent Semantic Analysis (LSA):
 - `n_components`
 - o not performing LSA
- Normalize the data/not normalize it with “Normalizer” (included in the notebook).

The questions to be responded in this part are:

- Which tips can you give about constructing a text clustering with k-means? What do you recommend to do? What do you recommend not to do?
- What is the best clustering you have obtained? The quality of the cluster is the degree of correspondence between real class and assigned cluster. For example:
 - If there are 2 clusters and cluster 0 contains all examples of positive class and cluster 1 contains all examples of negative class, the clustering is perfect.
 - If there are 2 clusters and cluster 1 contains all examples of positive class and cluster 0 contains all examples of negative class, the clustering is also perfect.
 - If there are 2 clusters and cluster 0 contains 50% of examples of positive class and 50% of examples of negative class, and statistics in cluster 1 are similar, the clustering quality is the worst possible.

3.1 Strategy (tips)

We use of course *HGS* to compute the best parameters for k-means.

But first we transformed the prediction of k-means clustering as they come from a binary supervised classifier, by means of the following method:

```
def get_prediction_from_cluster(X, pipeline):  
    """ Transform cluster assignment in y_pred object """  
    def swap_label(label):  
        if label == 1:
```



```

        return '0'
    elif label == 0:
        return '1'
    else:
        return str(label)
labels = pipeline.predict(X_test)
labels_predicted = [str(label) for label in labels]
predicted = pd.Series(labels_predicted)
accuracy = metrics.accuracy_score(y_test, predicted)
labels_predicted_reverse = [swap_label(label) for label in labels]
predicted_reverse = pd.Series(labels_predicted_reverse)
accuracy_reverse = metrics.accuracy_score(y_test, predicted_reverse)
if accuracy_reverse > accuracy: predicted = predicted_reverse
return predicted

```

The basic idea is to compute the accuracy assigning arbitrarily the label 1 and 0 to one of the cluster, and then the reciprocal assignment. The choice is the one with the best computed accuracy. To do so, we compute k-means with two clusters, in order to simplify the binary class assignment.

With this strategy, we include k-means as a classifier in our previous pipeline.

One note about the metrics

In all this work we use the *micro avg* metrics from sklearn classification report: precision, recall, f1-score. But, for a binary classifier all three metrics are equal and equivalent to accuracy.

The heat map is computed for the precision metric.

3.2 Best cluster

3.3 Main process with prefixed parameters

Script 3.3.1 (python)

```

1 param_ini = {
2     'vect__smooth_idf': True,
3     'vect__sublinear_tf': True,
4     'vect__max_features': 500,
5     'vect__min_df': 1,
6     'vect__max_df': 1.,
7     'vect__stop_words': 'english',
8     'vect__strip_accents' : 'unicode',
9     'vect__analyzer' : 'word',
10    'vect__ngram_range' : (1, 2),
11    'vect__norm': 'l2',
12    'red_svd__n_components': 10,
13    'clf_knn__n_neighbors' : 2,
14    'clf_kmeans__n_clusters' : 2,
15    'red_kbest__k' : 3,
16    'red_percentile__score_func' : f_classif,
17    'red_percentile__percentile' : 10,
18    'scaler': None
19    #'scaler__with_mean' : False
20 }

```

```

21
22 reducers=REDUCERS
23 classifiers=['kmeans']
24 df_metrics_fixed_kmeans = process_classifications(X_train, y_train, X_test, y_test,\
25                                                    param_ini, reducers=reducers,
                                                    ↪ classifiers=classifiers)

```

Output

```

### Reducer: svd    Classifier: kmeans
Accuracy 0.9320208023774146

### Reducer: kbest  Classifier: kmeans
Accuracy 0.7964338781575037

### Reducer: percentile  Classifier: kmeans
Accuracy 0.9071322436849926

### Reducer: none   Classifier: kmeans
Accuracy 0.9468796433878157

```

Script 3.3.2 (python)

```

1 display(df_metrics_fixed_kmeans.iloc[:, :-1])

```

	reducer	classifier	precision	recall	f1-score	support
0	svd	kmeans	0.932021	0.932021	0.932021	2692
1	kbest	kmeans	0.796434	0.796434	0.796434	2692
2	percentile	kmeans	0.907132	0.907132	0.907132	2692
3	none	kmeans	0.946880	0.946880	0.946880	2692

3.4 Main process with grid search parameters

Script 3.4.1 (python)

```

1 eng_and_custom_stopwords = improve_stop_words(X_train, 200)
2 # First we adjust the vectorizer parameters
3 param_grid_vectorizer = {
4     'vect__norm': ['l1', 'l2', None],
5     'vect__max_features': [500, 1000],
6     'vect__min_df': [0.0],
7     'vect__max_df': [1.],
8     'vect__stop_words': [None, 'english', eng_and_custom_stopwords]
9 }
10
11 # Then we adjust the reducer parameters
12 param_grid_reducers = [
13     {

```

```

14     'red_svd__n_components' : [2, 3, 10, 30, 40, 100],
15     'red_kbest__k' : [10, 20, 50],
16     'red_percentile__score_func' : [f_classif],
17     'red_percentile__percentile' : [5, 10]
18 }
19 ]
20
21 # Then we adjust the classifier parameters
22 param_grid_classifiers = [{
23     'clf_kmeans__n_clusters' : [2]
24 },
25 {
26     'clf_kmeans__norm': [None]
27 }
28 ]
29
30 df_metrics_all_kmeans = hyper_grid_search([param_grid_vectorizer, param_grid_reducers,
31     ↪ param_grid_classifiers],\
                                     df_metrics_fixed_kmeans, reducers=reducers,
                                     ↪ classifiers=classifiers)

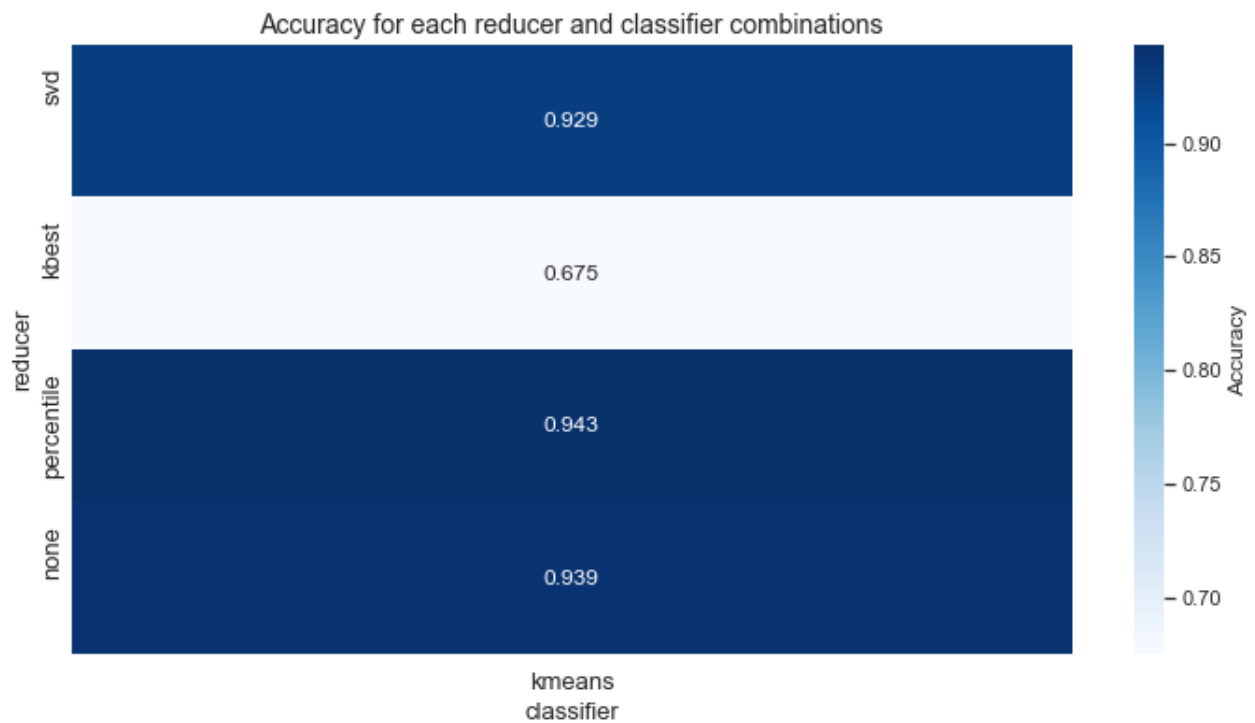
```

Script 3.4.2 (python)

```

1 plot_heatmap(df_metrics_all_kmeans, cols=['reducer', 'classifier', 'precision'],
  ↪ figsize=(12,6))
2 display(df_metrics_all_kmeans.iloc[:,-1])

```



	reducer	classifier	precision	recall	f1-score	support
0	svd	kmeans	0.928678	0.928678	0.928678	2692
0	kbest	kmeans	0.674963	0.674963	0.674963	2692
0	percentile	kmeans	0.943165	0.943165	0.943165	2692
0	none	kmeans	0.939079	0.939079	0.939079	2692

Script 3.4.3 (python)

```

1 # Improve reducers
2 param_grid_reducers = [
3     {
4         'red_kbest__k' : [3, 5, 8, 10],
5         'red_percentile__score_func' : [f_classif],
6         'red_percentile__percentile' : [1, 2, 3]
7     }
8 ]
9
10 df_metrics_all_kmeans_2 = hyper_grid_search([param_grid_reducers],
11                                             df_metrics_all_kmeans, reducers=reducers,
12                                             ↪ classifiers=classifiers)

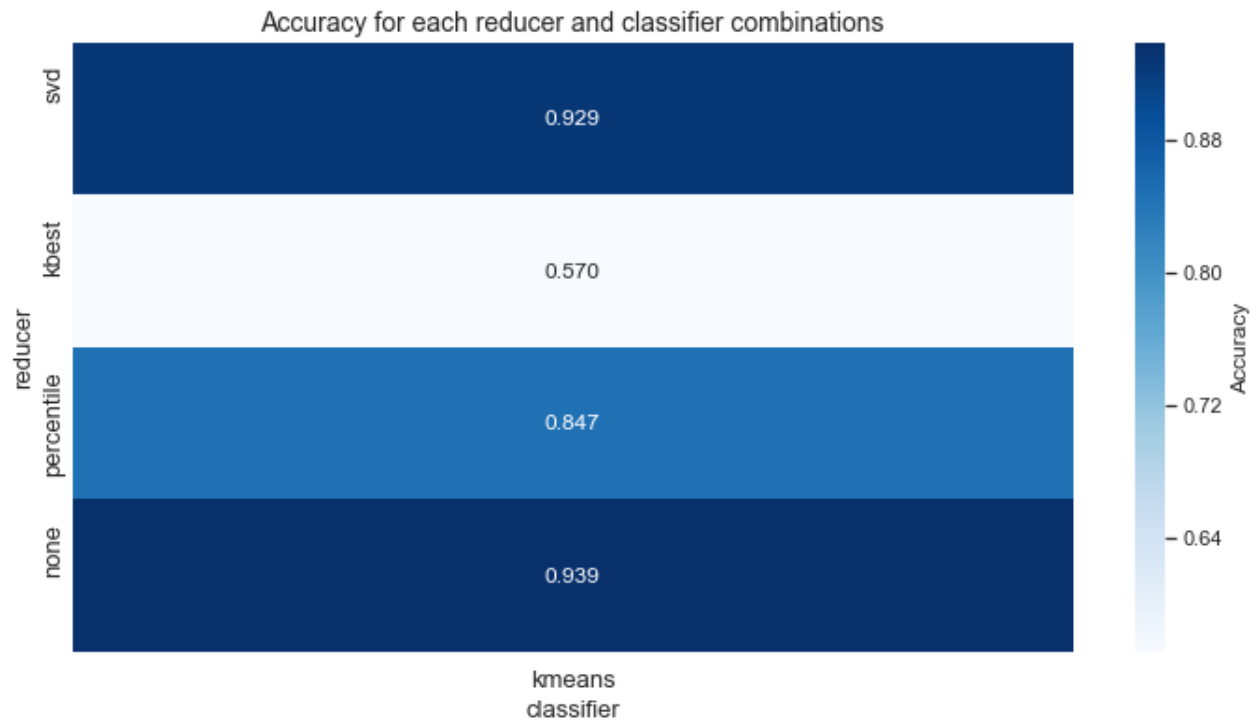
```

Script 3.4.4 (python)

```

1 plot_heatmap(df_metrics_all_kmeans_2, cols=['reducer', 'classifier', 'precision'],
2             ↪ figsize=(12,6))
3 display(df_metrics_all_kmeans_2.iloc[:, :-1])

```



	reducer	classifier	precision	recall	f1-score	support
0	svd	kmeans	0.928678	0.928678	0.928678	2692
0	kbest	kmeans	0.570208	0.570208	0.570208	2692
0	percentile	kmeans	0.846954	0.846954	0.846954	2692
0	none	kmeans	0.939079	0.939079	0.939079	2692

3.5 Reference process

Script 3.5.1 (python)

```

1 from sklearn.cluster import KMeans
2 from sklearn.metrics import calinski_harabaz_score
3 from sklearn.preprocessing import Normalizer
4 from sklearn.pipeline import make_pipeline
5 from sklearn.preprocessing import Normalizer
6
7 def get_X_transform(X):
8     vectorizador = TfidfVectorizer(max_df=1., max_features=1000, norm='l2',
9                                   min_df=1, stop_words='english',
10                                  #stop_words=stopwords,
11                                  #token_pattern=r'(?u)\b[A-Za-z]+\b',
12                                  #token_pattern=r'(?ui)\b\w*[a-z]+\w*\b',
13                                  use_idf=True)
14

```

```

15 vectorizador = TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
16                               encoding='utf-8', input='content',
17                               lowercase=True, max_df=1.0, max_features=1000, min_df=1,
18                               ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
19                               stop_words='english', strip_accents='unicode', sublinear_tf=True,
20                               token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
21                               vocabulary=None)
22
23 X = vectorizador.fit_transform(X)
24
25 print(X.shape)
26 n_componentes = 100
27 svd_truncado = TruncatedSVD(n_componentes)
28 normalizador = Normalizer(copy=False)
29
30 lsa = make_pipeline(svd_truncado, normalizador)
31 #lsa = svd_truncado
32
33 X_lsa = lsa.fit_transform(X)
34
35 varianza_explicada = svd_truncado.explained_variance_ratio_.sum()
36 normalizer = Normalizer()
37 X_lsa_norm = normalizer.fit_transform(X_lsa)
38 return X_lsa_norm
39
40 X_km = get_X_transform(X_train)
41 qmetric = calinski_harabaz_score
42
43 Nclusters_max = 15
44 Nrepetitions = 100
45
46 qualities = []
47 inertias = []
48 models = []
49 kini = 1
50 kfin = 20
51 for k in range(kini, kfin+1):
52     print("Evaluando k=%d" % k)
53     km = KMeans(n_clusters=k,
54                 init='k-means++', n_init=Nrepetitions,
55                 max_iter=500, random_state=2)
56     km.fit(X_km)
57     models.append(km)
58     inertias.append(km.inertia_)
59     if k > 1:
60         qualities.append(qmetric(X_km, km.labels_))
61         #qualities.append(km.score(X_km))
62     else:
63         qualities.append(0)

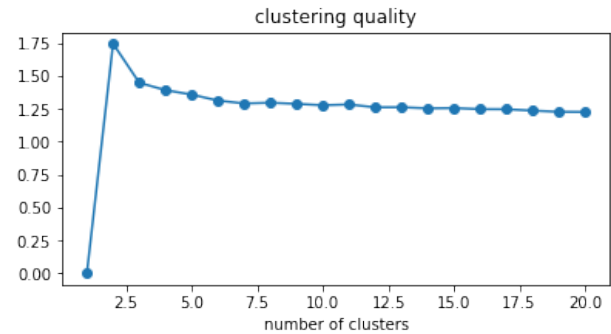
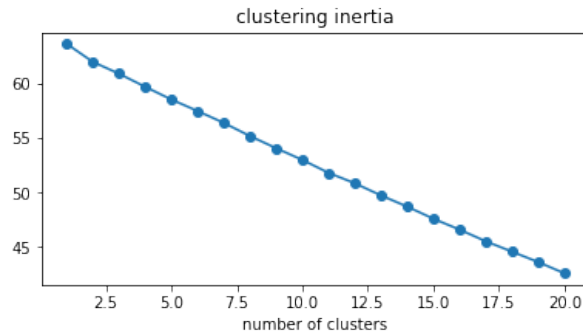
```

Output

```
(67, 1000)
Evaluando k=1
Evaluando k=2
Evaluando k=3
Evaluando k=4
Evaluando k=5
Evaluando k=6
Evaluando k=7
Evaluando k=8
Evaluando k=9
Evaluando k=10
Evaluando k=11
Evaluando k=12
Evaluando k=13
Evaluando k=14
Evaluando k=15
Evaluando k=16
Evaluando k=17
Evaluando k=18
Evaluando k=19
Evaluando k=20
```

Script 3.5.2 (python)

```
1 fig = plt.figure(figsize=(14,3))
2
3 ax = plt.subplot(1,2,1)
4 plt.plot(range(kini,kfin+1), inertias, marker='o')
5 plt.xlabel('number of clusters')
6 plt.title('clustering inertia')
7
8 ax = plt.subplot(1,2,2)
9 plt.plot(range(kini,kfin+1), qualities, marker='o')
10 plt.xlabel('number of clusters')
11 plt.title('clustering quality')
12 plt.show()
13
14 best = pd.Series(qualities).idxmax() # get index for the best model
15 print("Best number of clusters", best)
16 km = models[best]
17 n_clusters = km.get_params()['n_clusters']
18 clusters = km.labels_
19 print ('Number of clusters of best quality', n_clusters)
```



Output

Best number of clusters 1
Number of clusters of best quality 2

Script 3.5.3 (python)

```
1 # We choose the best option to evaluate the quality of prediction
2 X = X_test
3 y = y_test
4 X_km = get_X_transform(X)
5 labels = km.fit_predict(X_km)
6 #print(labels)
7 # First we try with labels as is
8 labels_predicted = [str(label) for label in labels]
9 predicted = pd.Series(labels_predicted)
10 #print(labels_predicted)
11 print(metrics.classification_report(y, predicted))
12 print(metrics.confusion_matrix(y, predicted))
13
14 # Alternatively we invert the label to match the real labels of each group
15 labels_predicted = [str((label + 1)%2) for label in labels]
16 #print(labels_predicted)
17 predicted = pd.Series(labels_predicted)
18 print(metrics.classification_report(y, predicted))
19 print(metrics.confusion_matrix(y, predicted))
```

Output

```
(33, 1000)
      precision    recall  f1-score   support

     0       1.00      0.87      0.93         15
     1       0.90      1.00      0.95         18

   micro avg       0.94      0.94      0.94         33
   macro avg       0.95      0.93      0.94         33
  weighted avg       0.95      0.94      0.94         33
```



```

[[13  2]
 [ 0 18]]

```

	precision	recall	f1-score	support
0	0.10	0.13	0.11	15
1	0.00	0.00	0.00	18
micro avg	0.06	0.06	0.06	33
macro avg	0.05	0.07	0.06	33
weighted avg	0.05	0.06	0.05	33

```

[[ 2 13]
 [18  0]]

```