Text Mining Assignment

Elena Montenegro, Fernando Freire

May 2, 2019

Contents

1 Modules importation and data loading

Script 1.0.1 (python)

```python
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sys
%matplotlib inline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

from sklearn.naive_bayes import MultinomialNB
from sklearn.decomposition import TruncatedSVD# SVD = Singular Value Descomposition
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler, Normalizer, MinMaxScaler, MaxAbsScaler
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.svm import SVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.feature_extraction import stop_words
from sklearn.base import TransformerMixin
from sklearn.cluster import KMeans
from sklearn.metrics import calinski_harabaz_score, accuracy_score
from sklearn.preprocessing import Normalizer, LabelBinarizer, OneHotEncoder
from sklearn.metrics import make_scorer

random_state=0
```

Script 1.0.2 (python)

```python
# Data loading
#NROWS = sys.maxsize
NROWS = 50
## Negative dataset
df_neg = pd.read_csv('./practica_clase/PRECISION_MEDICINE/negative_training_abstracts.tsv',
 ↪  sep='\t',
                     header=None, nrows = NROWS)

df_neg.columns = ['Accession number', 'Title', 'Abstract']
df_neg['Label'] = '0' #'neg'

display(df_neg.head())
```

```

corpus_neg = list(df_neg['Abstract'].values)
### len(corpus_neg) # 4078

## Positive
df_pos = pd.read_csv('./practica_clase/PRECISION_MEDICINE/positive_training_abstracts.tsv',
↪  sep='\t',
                     header=None, nrows = NROWS)

df_pos.columns = ['Accession number', 'Title', 'Abstract']
df_pos['Label'] = '1' # 'pos'
display(df_pos.head())

# Add corpus
df_corpus = df_neg.append(df_pos)
display(df_corpus.head())

# len(corpus) # 8156

labels = df_corpus['Label']
corpus = df_corpus['Abstract']
# len(labels) # 8156

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 Orai1 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

```
100 100
```

1.1 Data split

Script 1.1.1 (python)

```python
TEST_SIZE = 0.33
X_train, X_test, y_train, y_test = train_test_split(
    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 thesequestions:

- 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 Pipelines

2.1.1 Find additional stopwords

Script 2.1.1 (python)

```python
def get_top_n_words(corpus, n=None):
    """
    List the top n words in a vocabulary according to occurrence in a text corpus.
    """
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)

    return words_freq[:n]

def improve_stop_words(X_train, n=50):
    """
    """
    common_words = [i[0] for i in get_top_n_words(X_train, n)]
    eng_and_custom_stopwords = set(list(stop_words.ENGLISH_STOP_WORDS) + common_words)
    print(len(eng_and_custom_stopwords))
    return eng_and_custom_stopwords
```

2.1.2 Pipelining methods

Script 2.1.2 (python)

```
CLASSIFIERS = ['knn', 'dtree', 'nb', 'lr', 'svc', 'lsvc']
CLASSIFIERS_UNSUPERVISED = ['kmeans']
REDUCERS = ['svd', 'kbest', 'percentile', None]
CV = 4
VERBOSE = False

def create_text_pipeline(reducer='svd', classifier="nb"):
    """ Create text vectorization pipeline with optional dimensionality reduction"""
    assert reducer in REDUCERS, "ERROR: Reducer %s not supported, only %s" % (reducer,
    ↪  REDUCERS)
    assert classifier in CLASSIFIERS + CLASSIFIERS_UNSUPERVISED,\
        "ERROR: Classifier %s not supported, only %s" % (classifier, CLASSIFIERS +
        ↪  CLASSIFIERS_UNSUPERVISED)
    pipeline = [
        ('vect', TfidfVectorizer()),
        ('scaler', StandardScaler())
    ]
    # Reduce dimensions
    if reducer == 'svd':
        pipeline.append(('red_svd', TruncatedSVD()))
    elif reducer == 'kbest':
        pipeline.append(('red_kbest', SelectKBest()))
    elif reducer == 'percentile':
        pipeline.append(('red_percentile', SelectPercentile()))
    elif reducer == None:
        pass

    # Classify
    if classifier == "nb":
        if reducer == 'svd':
            pipeline.append(('mm_scaler', MinMaxScaler()))
        elif reducer == 'kbest':
            pipeline.append(('mm_scaler', MaxAbsScaler()))
        elif reducer == 'percentile':
            pipeline.append(('mm_scaler', MaxAbsScaler()))
        elif reducer == None:
            pass
        pipeline.append(('clf_' + classifier, MultinomialNB()))
    elif classifier == "lr":
        pipeline.append(('clf_' + classifier, LogisticRegression()))
    elif classifier == "svc":
        pipeline.append(('clf_' + classifier, SVC()))
    elif classifier == "lsvc":
        pipeline.append(('clf_' + classifier, LinearSVC()))
    elif classifier == "dtree":
        pipeline.append(('clf_' + classifier, DecisionTreeClassifier()))
    elif classifier == "knn":
        pipeline.append(('clf_' + classifier, KNeighborsClassifier()))
    elif classifier == "kmeans":
```

```
        pipeline.append(('norm', Normalizer()))
        pipeline.append(('cluster_kmeans', KMeans()))
    elif classifier == None:
        pass

    return Pipeline(pipeline)

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

def get_filtered_params(parameters, pipeline):
    """ Filter the params that aren't related to steps in the pipeline """
    filtered_params = {}
    for param_key in parameters.keys():
        if param_key.split('__')[0] in pipeline.named_steps.keys():
            filtered_params[param_key] = parameters[param_key]
    return filtered_params

def get_filtered_set(parameters, pipeline):
    """ Filter the params that aren't related to steps in the pipeline """
    if type(parameters) == dict:
        return get_filtered_params(parameters, pipeline)
    else:
        filtered_set = []
        for param_set in parameters:
            filtered_set.append(get_filtered_params(param_set, pipeline))
        return filtered_set

def prediction_metrics(X_train, y_train, X_test, y_test, parameters, results, reducer="svd",
↪   classifier="nb"):
    """
    """
    print("### Reducer: %s   Classifier: %s" %(reducer, classifier))
    pipeline = create_text_pipeline(reducer=reducer, classifier=classifier)
    pipeline.set_params(**get_filtered_params(parameters, pipeline))
    if VERBOSE: print("Pipeline", pipeline.named_steps)
```

```
    pipeline.fit(X_train, y_train)
    if classifier in CLASSIFIERS_UNSUPERVISED:
        predicted = get_prediction_from_cluster(X_test, pipeline)
    else:
        predicted = pipeline.predict(X_test)
    print()
    accuracy = metrics.accuracy_score(y_test, predicted)
    print("Accuracy", accuracy)
    clf_rep = metrics.classification_report(y_test, predicted, output_dict=True, digits=2)
    if VERBOSE: print(clf_rep['micro avg'])
    if VERBOSE: print(metrics.confusion_matrix(y_test, predicted))

    results.append([reducer, classifier, accuracy] + list(clf_rep['micro avg'].values()))

def process_classifications(X_train, y_train, X_test, y_test, parameters,
                            classifiers=CLASSIFIERS, reducers=REDUCERS):
    """
    """
    results = []
    for classifier in classifiers:
        for reducer in reducers:
            prediction_metrics(X_train, y_train, X_test, y_test, parameters, results,
             ↪  reducer, classifier)
    # Group all results into a dataframe
    df = pd.DataFrame(results, columns=['reducer', 'classifier', 'accuracy', 'precision',
     ↪  'recall', 'f1-score', 'support'])
    df['classifier'].fillna('None',inplace=True)

    return df

def prediction_metrics_grid(X_train, y_train, X_test, y_test, parameters_grid, results=[],
                            reducer="svd", classifier="nb", cv=CV):
    """
    """
    print("### Reducer: %s   Classifier: %s" %(reducer, classifier))
    pipeline = create_text_pipeline(reducer=reducer, classifier=classifier)
    filtered_params = get_filtered_set(parameters_grid, pipeline)
    #scoring = {'accuracy': make_scorer(accuracy_score), 'calinski':
     ↪  make_scorer(calinski_harabaz_score)}
    scoring = {'accuracy': make_scorer(accuracy_score)}
    grid_model = GridSearchCV(pipeline, filtered_params, cv=cv, iid=False, error_score=0,
                              scoring=None, refit=False)
    grid_model.fit(X_train, y_train)
    print()
    print("Best parameters")
    for param_name in sorted(grid_model.best_params_.keys()):
        print("\t%s: %r" % (param_name, grid_model.best_params_[param_name]))
    pipeline.set_params(**grid_model.best_params_)
    pipeline.fit(X_train, y_train)
    if classifier in CLASSIFIERS_UNSUPERVISED:
        predicted = get_prediction_from_cluster(X_test, pipeline)
    else:
```

```
        predicted = pipeline.predict(X_test)
    print()
    accuracy = metrics.accuracy_score(y_test, predicted)
    print("Accuracy", accuracy)
    clf_rep = metrics.classification_report(y_test, predicted, output_dict=True, digits=2)
    if VERBOSE: print(clf_rep['micro avg'])
    if VERBOSE: print(metrics.confusion_matrix(y_test, predicted))

    results.append([reducer, classifier, accuracy] + list(clf_rep['micro avg'].values()))


def process_classifications_grid(X_train, y_train, X_test, y_test, parameters, cv=CV,
                            classifiers=CLASSIFIERS, reducers=REDUCERS):
    """
    """
    results = []
    for classifier in classifiers:
        for reducer in reducers:
            prediction_metrics_grid(X_train, y_train, X_test, y_test, parameters,
                                    results, reducer, classifier, cv=cv)
    # Group all results into a dataframe
    df = pd.DataFrame(results, columns=['reducer', 'classifier', 'accuracy', 'precision',
    ↪  'recall', 'f1-score', 'support'])
    df['classifier'].fillna('None',inplace=True)

    return df
```

2.2 Main process with prefixed parameters

Script 2.2.1 (python)

```
VERBOSE = False
# First set of parameters
param_set_1 = {
    'vect__norm': None,
    'vect__smooth_idf': True,
    'vect__sublinear_tf': True,
    'vect__max_features': 1000,
    'vect__min_df': 6,
    'vect__stop_words': 'english',
    'vect__strip_accents' : 'unicode',
    'vect__analyzer' : 'word',
    'vect__token_pattern': r'\w{1,}',
    'vect__ngram_range' : (1, 2),
    'scaler' : None,
    'red_kbest__k' : 3,
    'red_percentile__score_func' : f_classif,
    'red_percentile__percentile' : 10,
    #'scaler__with_mean' : False,
    'vect__norm': 'l2',
```

```
    'red_svd__n_components': 40,
    'clf_knn__n_neighbors' : 2,
}

# More stop words
eng_and_custom_stopwords = improve_stop_words(X_train, 200)
param_set_1['vect__stop_words'] = eng_and_custom_stopwords

data_classifier = process_classifications(X_train, y_train, X_test, y_test, param_set_1,
                            reducers=REDUCERS, classifiers=CLASSIFIERS)

#process_classifications(X_train, y_train, X_test, y_test, param_set_1)
```

Output

```
462
### Reducer: svd   Classifier: knn

Accuracy 0.6666666666666666
### Reducer: kbest   Classifier: knn

Accuracy 0.7272727272727273
### Reducer: percentile   Classifier: knn

Accuracy 0.6666666666666666
### Reducer: None   Classifier: knn

Accuracy 0.6363636363636364
### Reducer: svd   Classifier: dtree

Accuracy 0.6363636363636364
### Reducer: kbest   Classifier: dtree

Accuracy 0.6060606060606061
### Reducer: percentile   Classifier: dtree

Accuracy 0.696969696969697
### Reducer: None   Classifier: dtree

Accuracy 0.42424242424242425
### Reducer: svd   Classifier: nb

Accuracy 0.8484848484848485
### Reducer: kbest   Classifier: nb

Accuracy 0.6666666666666666
### Reducer: percentile   Classifier: nb

Accuracy 0.7575757575757576
### Reducer: None   Classifier: nb
```

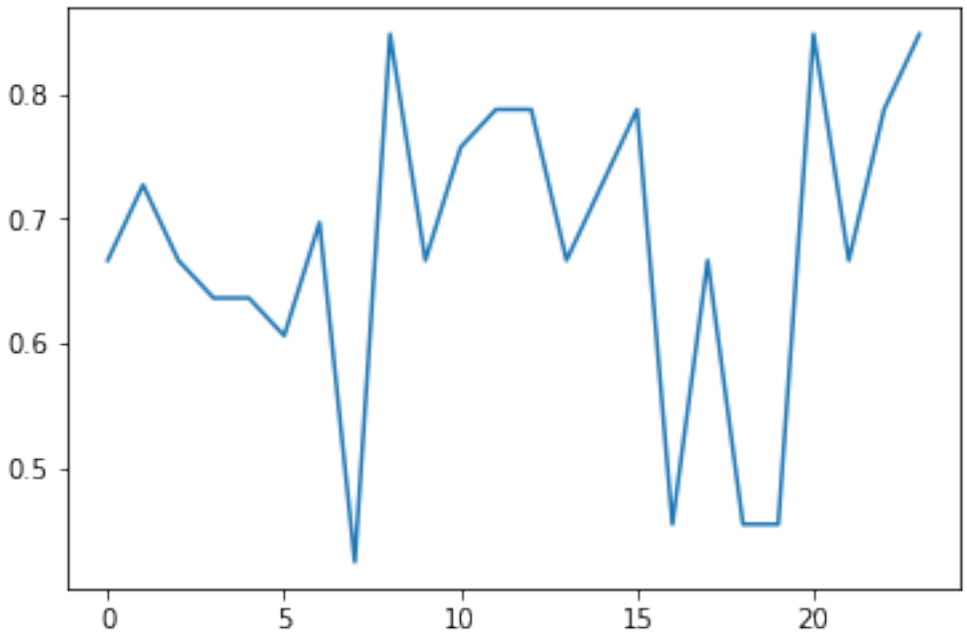
```
Accuracy 0.7878787878787878
### Reducer: svd   Classifier: lr

Accuracy 0.7878787878787878
### Reducer: kbest   Classifier: lr

Accuracy 0.6666666666666666
### Reducer: percentile   Classifier: lr

Accuracy 0.7272727272727273
### Reducer: None   Classifier: lr

Accuracy 0.7878787878787878
### Reducer: svd   Classifier: svc

Accuracy 0.45454545454545453
### Reducer: kbest   Classifier: svc

Accuracy 0.6666666666666666
### Reducer: percentile   Classifier: svc

Accuracy 0.45454545454545453
### Reducer: None   Classifier: svc

Accuracy 0.45454545454545453
### Reducer: svd   Classifier: lsvc

Accuracy 0.8484848484848485
### Reducer: kbest   Classifier: lsvc

Accuracy 0.6666666666666666
### Reducer: percentile   Classifier: lsvc

Accuracy 0.7878787878787878
### Reducer: None   Classifier: lsvc

Accuracy 0.8484848484848485
```

Script 2.2.2 (python)

```python
display(data_classifier)
plt.plot(data_classifier['accuracy'].values);
```

	reducer	classifier	accuracy	precision	recall	f1-score	support
0	svd	knn	0.666667	0.666667	0.666667	0.666667	33
1	kbest	knn	0.727273	0.727273	0.727273	0.727273	33
2	percentile	knn	0.666667	0.666667	0.666667	0.666667	33
3	None	knn	0.636364	0.636364	0.636364	0.636364	33
4	svd	dtree	0.636364	0.636364	0.636364	0.636364	33
5	kbest	dtree	0.606061	0.606061	0.606061	0.606061	33
6	percentile	dtree	0.696970	0.696970	0.696970	0.696970	33

7	None	dtree	0.424242	0.424242	0.424242	0.424242	33
8	svd	nb	0.848485	0.848485	0.848485	0.848485	33
9	kbest	nb	0.666667	0.666667	0.666667	0.666667	33
10	percentile	nb	0.757576	0.757576	0.757576	0.757576	33
11	None	nb	0.787879	0.787879	0.787879	0.787879	33
12	svd	lr	0.787879	0.787879	0.787879	0.787879	33
13	kbest	lr	0.666667	0.666667	0.666667	0.666667	33
14	percentile	lr	0.727273	0.727273	0.727273	0.727273	33
15	None	lr	0.787879	0.787879	0.787879	0.787879	33
16	svd	svc	0.454545	0.454545	0.454545	0.454545	33
17	kbest	svc	0.666667	0.666667	0.666667	0.666667	33
18	percentile	svc	0.454545	0.454545	0.454545	0.454545	33
19	None	svc	0.454545	0.454545	0.454545	0.454545	33
20	svd	lsvc	0.848485	0.848485	0.848485	0.848485	33
21	kbest	lsvc	0.666667	0.666667	0.666667	0.666667	33
22	percentile	lsvc	0.787879	0.787879	0.787879	0.787879	33
23	None	lsvc	0.848485	0.848485	0.848485	0.848485	33



2.3 Main process with grid search parameters

Script 2.3.1 (python)

```python
parameters_grid = {
    'vect__norm': ['l1', 'l2', None],
    'vect__smooth_idf': [True],
    'vect__sublinear_tf': [True],
    'vect__max_features': [900, 1000],
    'vect__min_df': [1, 5, 6],
    'vect__max_df': [1., 5., 6],
    'vect__stop_words': [None, 'english', eng_and_custom_stopwords],
    'vect__strip_accents' : ['unicode'],
    'vect__analyzer' : ['word'],
    'vect__token_pattern': [r'\w{1,}'],
    'vect__ngram_range' : [(1, 2)],
    'scaler' : [None],
    'red_svd__n_components': [2, 30, 40],
    'clf_knn__n_neighbors' : [2, 5],
    'red_percentile__score_func' : [f_classif],
    'red_percentile__percentile' : [10],
    'red_kbest__k' : [3]
}

eng_and_custom_stopwords = improve_stop_words(X_train, 200)
#prediction_metrics_grid(X_train, y_train, X_test, y_test, parameters_grid, reducer='svd',
↪  classifier="knn", cv=2)
data_classifier_grid = process_classifications_grid(X_train, y_train, X_test, y_test,
↪  parameters_grid, cv=3)
```

Output

```
462
### Reducer: svd   Classifier: knn

Best parameters
        clf_knn__n_neighbors: 5
        red_svd__n_components: 2
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        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
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9090909090909091
```

```
### Reducer: kbest   Classifier: knn

Best parameters
        clf_knn__n_neighbors: 2
        red_kbest__k: 3
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 6
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: percentile   Classifier: knn

Best parameters
        clf_knn__n_neighbors: 5
        red_percentile__percentile: 10
        red_percentile__score_func: <function f_classif at 0x1a1b98f1e0>
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.8787878787878788
### Reducer: None   Classifier: knn

Best parameters
        clf_knn__n_neighbors: 5
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l2'
        vect__smooth_idf: True
        vect__stop_words: 'english'
```

```
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9090909090909091
### Reducer: svd   Classifier: dtree

Best parameters
        red_svd__n_components: 2
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        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
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: kbest   Classifier: dtree

Best parameters
        red_kbest__k: 3
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: percentile   Classifier: dtree

Best parameters
        red_percentile__percentile: 10
        red_percentile__score_func: <function f_classif at 0x1a1b98f1e0>
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 6
        vect__max_features: 1000
        vect__min_df: 5
        vect__ngram_range: (1, 2)
```

```
        vect__norm: None
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.8787878787878788
### Reducer: None   Classifier: dtree

Best parameters
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l2'
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.8787878787878788
### Reducer: svd   Classifier: nb

Best parameters
        red_svd__n_components: 30
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 1
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9090909090909091
### Reducer: kbest   Classifier: nb

Best parameters
        red_kbest__k: 3
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 6
```

```
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.5454545454545454
### Reducer: percentile   Classifier: nb

Best parameters
        red_percentile__percentile: 10
        red_percentile__score_func: <function f_classif at 0x1a1b98f1e0>
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9090909090909091
### Reducer: None   Classifier: nb

Best parameters
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l2'
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9090909090909091
### Reducer: svd   Classifier: lr

Best parameters
        red_svd__n_components: 30
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
```

```
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: None
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: kbest   Classifier: lr

Best parameters
        red_kbest__k: 3
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 1
        vect__ngram_range: (1, 2)
        vect__norm: None
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: percentile   Classifier: lr

Best parameters
        red_percentile__percentile: 10
        red_percentile__score_func: <function f_classif at 0x1a1b98f1e0>
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        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
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: None   Classifier: lr

Best parameters
        scaler: None
```

```
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l2'
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: svd   Classifier: svc

Best parameters
        red_svd__n_components: 2
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 5.0
        vect__max_features: 900
        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
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: kbest   Classifier: svc

Best parameters
        red_kbest__k: 3
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 1
        vect__ngram_range: (1, 2)
        vect__norm: None
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9393939393939394
### Reducer: percentile   Classifier: svc

Best parameters
```

```
        red_percentile__percentile: 10
        red_percentile__score_func: <function f_classif at 0x1a1b98f1e0>
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 6
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: None
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.8181818181818182
### Reducer: None   Classifier: svc

Best parameters
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 6
        vect__ngram_range: (1, 2)
        vect__norm: None
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.8787878787878788
### Reducer: svd   Classifier: lsvc

Best parameters
        red_svd__n_components: 30
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: None
        vect__smooth_idf: True
        vect__stop_words: 'english'
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.8787878787878788
### Reducer: kbest   Classifier: lsvc
```

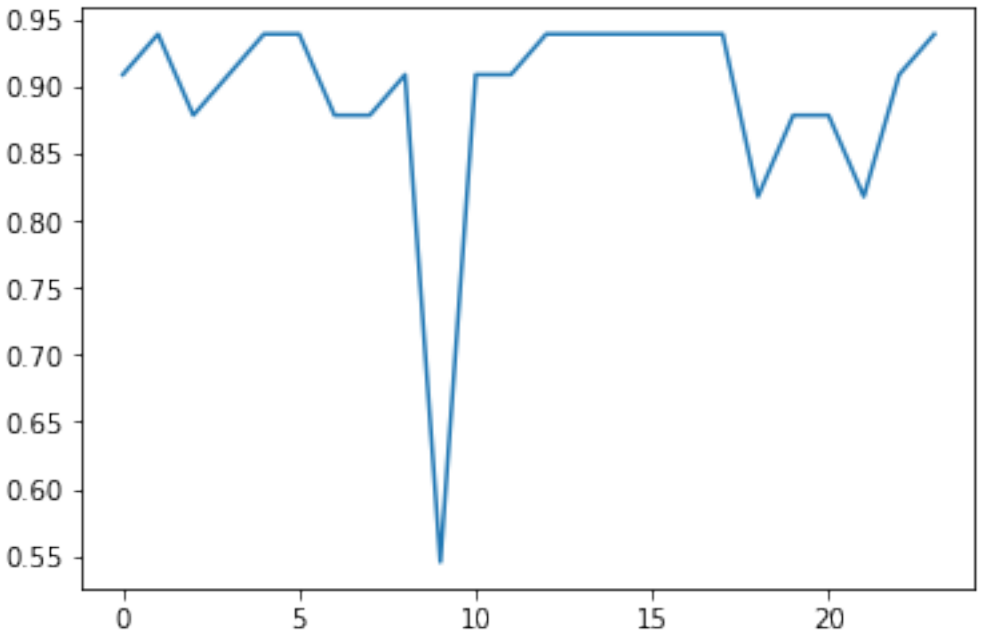
```
Best parameters
        red_kbest__k: 3
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 1
        vect__ngram_range: (1, 2)
        vect__norm: 'l2'
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.8181818181818182
### Reducer: percentile   Classifier: lsvc

Best parameters
        red_percentile__percentile: 10
        red_percentile__score_func: <function f_classif at 0x1a1b98f1e0>
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 6
        vect__max_features: 900
        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
        vect__token_pattern: '\\w{1,}'

Accuracy 0.9090909090909091
### Reducer: None   Classifier: lsvc

Best parameters
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 900
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l2'
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'
```

```
Accuracy 0.9393939393939394
```

Script 2.3.2 (python)

```python
display(data_classifier_grid)
plt.plot(data_classifier_grid['accuracy'].values);
```

	reducer	classifier	accuracy	precision	recall	f1-score	support
0	svd	knn	0.909091	0.909091	0.909091	0.909091	33
1	kbest	knn	0.939394	0.939394	0.939394	0.939394	33
2	percentile	knn	0.878788	0.878788	0.878788	0.878788	33
3	None	knn	0.909091	0.909091	0.909091	0.909091	33
4	svd	dtree	0.939394	0.939394	0.939394	0.939394	33
5	kbest	dtree	0.939394	0.939394	0.939394	0.939394	33
6	percentile	dtree	0.878788	0.878788	0.878788	0.878788	33
7	None	dtree	0.878788	0.878788	0.878788	0.878788	33
8	svd	nb	0.909091	0.909091	0.909091	0.909091	33
9	kbest	nb	0.545455	0.545455	0.545455	0.545455	33
10	percentile	nb	0.909091	0.909091	0.909091	0.909091	33
11	None	nb	0.909091	0.909091	0.909091	0.909091	33
12	svd	lr	0.939394	0.939394	0.939394	0.939394	33
13	kbest	lr	0.939394	0.939394	0.939394	0.939394	33
14	percentile	lr	0.939394	0.939394	0.939394	0.939394	33
15	None	lr	0.939394	0.939394	0.939394	0.939394	33
16	svd	svc	0.939394	0.939394	0.939394	0.939394	33
17	kbest	svc	0.939394	0.939394	0.939394	0.939394	33
18	percentile	svc	0.818182	0.818182	0.818182	0.818182	33
19	None	svc	0.878788	0.878788	0.878788	0.878788	33
20	svd	lsvc	0.878788	0.878788	0.878788	0.878788	33
21	kbest	lsvc	0.818182	0.818182	0.818182	0.818182	33
22	percentile	lsvc	0.909091	0.909091	0.909091	0.909091	33
23	None	lsvc	0.939394	0.939394	0.939394	0.939394	33



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 Main process with prefixed parameters

Script 3.1.1 (python)

```python
param_set_1 = {
    '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__token_pattern': r'\w{1,}',
    'vect__ngram_range' : (1, 2),
    #'scaler__with_mean' : False,
    'vect__norm': 'l2',
    'red_svd__n_components': 100,
    'clf_knn__n_neighbors' : 2,
    'cluster_kmeans__n_clusters' : 2,
    'red_kbest__k' : 3,
    'red_percentile__score_func' : f_classif,
    'red_percentile__percentile' : 10,
    'scaler': None,
    'norm': None
}

process_classifications(X_train, y_train, X_test, y_test, param_set_1, reducers=['svd'],
 ↪  classifiers=['kmeans'])
```

Output

```
### Reducer: svd   Classifier: kmeans
Pipeline {'vect': TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=1000, min_df=1,
        ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words='english', strip_accents='unicode', sublinear_tf=True,
        token_pattern='\\w{1,}', tokenizer=None, use_idf=True,
        vocabulary=None), 'scaler': None, 'red_svd': TruncatedSVD(algorithm='randomized',
         ↪  n_components=100, n_iter=5,
```

```
       random_state=None, tol=0.0), 'norm': None, 'cluster_kmeans': KMeans(algorithm='auto',
       ↪  copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)}

Accuracy 0.8787878787878788
              precision    recall  f1-score   support

           0       1.00      0.73      0.85        15
           1       0.82      1.00      0.90        18

   micro avg       0.88      0.88      0.88        33
   macro avg       0.91      0.87      0.87        33
weighted avg       0.90      0.88      0.88        33

[[11  4]
 [ 0 18]]
```

3.2 Main process with grid search parameters

Script 3.2.1 (python)

```python
eng_and_custom_stopwords = improve_stop_words(X_train, 200)
parameters_grid = [
    {'vect__norm': ['l1', 'l2', None],
    'vect__smooth_idf': [True],
    'vect__sublinear_tf': [True],
    'vect__max_features': [20, 30],
    'vect__min_df': [1, 5],
    'vect__max_df': [1., 6],
    'vect__stop_words': [None, 'english', eng_and_custom_stopwords],
    'vect__strip_accents' : ['unicode'],
    'vect__analyzer' : ['word'],
    'vect__token_pattern': [r'\w{1,}'],
    'vect__ngram_range' : [(1, 2)],
    'scaler' : [None],
    'red_svd__n_components': [2, 10, 15],
    'clf_knn__n_neighbors' : [2, 5],
    'cluster_kmeans__n_clusters' : [2],
    'norm' : [None]},
    # without svd
    {'vect__norm': ['l1', 'l2', None],
    'vect__smooth_idf': [True],
    'vect__sublinear_tf': [True],
    'vect__max_features': [20, 30],
    'vect__min_df': [1, 5],
    'vect__max_df': [1., 6],
    'vect__stop_words': [None, 'english', eng_and_custom_stopwords],
    'vect__strip_accents' : ['unicode'],
    'vect__analyzer' : ['word'],
```

```
    'vect__token_pattern': [r'\w{1,}'],
    'vect__ngram_range' : [(1, 2)],
    'scaler' : [None],
    'red_svd__n_components': [2, 10, 15],
    'clf_knn__n_neighbors' : [2, 5],
    'cluster_kmeans__n_clusters' : [2],
    'red__svd' : [None]}
]

eng_and_custom_stopwords = improve_stop_words(X_train, 200)
#prediction_metrics_grid(X_train, y_train, X_test, y_test, parameters_grid,
 ↪  reducer="reducer", classifier="kmeans", cv=CV)
process_classifications_grid(X_train, y_train, X_test, y_test, parameters_grid,
 ↪  classifiers=["kmeans"], cv=4)
```

Output

```
462
462
### Reducer: svd   Classifier: kmeans

Best parameters
        cluster_kmeans__n_clusters: 2
        norm: None
        red_svd__n_components: 2
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 30
        vect__min_df: 1
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.7878787878787878
              precision    recall  f1-score   support

           0       1.00      0.53      0.70        15
           1       0.72      1.00      0.84        18

   micro avg       0.79      0.79      0.79        33
   macro avg       0.86      0.77      0.77        33
weighted avg       0.85      0.79      0.77        33

[[ 8  7]
 [ 0 18]]
### Reducer: kbest   Classifier: kmeans
```

```
Best parameters
        cluster_kmeans__n_clusters: 2
        norm: None
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 30
        vect__min_df: 1
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.8181818181818182
              precision    recall  f1-score   support

           0       0.91      0.67      0.77        15
           1       0.77      0.94      0.85        18

   micro avg       0.82      0.82      0.82        33
   macro avg       0.84      0.81      0.81        33
weighted avg       0.83      0.82      0.81        33

[[10  5]
 [ 1 17]]
### Reducer: percentile   Classifier: kmeans

Best parameters
        cluster_kmeans__n_clusters: 2
        norm: None
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 20
        vect__min_df: 1
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.696969696969697
              precision    recall  f1-score   support

           0       0.86      0.40      0.55        15
           1       0.65      0.94      0.77        18
```

```
   micro avg       0.70      0.70      0.70        33
   macro avg       0.76      0.67      0.66        33
weighted avg       0.75      0.70      0.67        33

[[ 6  9]
 [ 1 17]]
### Reducer: None   Classifier: kmeans

Best parameters
        cluster_kmeans__n_clusters: 2
        norm: None
        scaler: None
        vect__analyzer: 'word'
        vect__max_df: 1.0
        vect__max_features: 20
        vect__min_df: 5
        vect__ngram_range: (1, 2)
        vect__norm: 'l1'
        vect__smooth_idf: True
        vect__stop_words: None
        vect__strip_accents: 'unicode'
        vect__sublinear_tf: True
        vect__token_pattern: '\\w{1,}'

Accuracy 0.7575757575757576
              precision    recall  f1-score   support

           0       0.89      0.53      0.67        15
           1       0.71      0.94      0.81        18

   micro avg       0.76      0.76      0.76        33
   macro avg       0.80      0.74      0.74        33
weighted avg       0.79      0.76      0.74        33

[[ 8  7]
 [ 1 17]]
```

3.3 Reference process

Script 3.3.1 (python)

```python
from sklearn.cluster import KMeans
from sklearn.metrics import calinski_harabaz_score
from sklearn.preprocessing import Normalizer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer

def get_X_transform(X):
    vectorizador = TfidfVectorizer(max_df=1., max_features=1000, norm='l2',
```

```
                                    min_df=1, stop_words='english',
                                    #stop_words=stopwords,
                                    #token_pattern=r'(?u)\b[A-Za-z]+\b',
                                    #token_pattern=r'(?ui)\b\w*[a-z]+\w*\b',
                                    use_idf=True)

    vectorizador = TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=1000, min_df=1,
        ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words='english', strip_accents='unicode', sublinear_tf=True,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)

    X = vectorizador.fit_transform(X)

    print(X.shape)
    n_componentes = 100
    svd_truncado = TruncatedSVD(n_componentes)
    normalizador = Normalizer(copy=False)

    lsa = make_pipeline(svd_truncado, normalizador)
    #lsa = svd_truncado

    X_lsa = lsa.fit_transform(X)

    varianza_explicada = svd_truncado.explained_variance_ratio_.sum()
    normalizer = Normalizer()
    X_lsa_norm = normalizer.fit_transform(X_lsa)
    return X_lsa_norm

X_km = get_X_transform(X_train)
qmetric = calinski_harabaz_score

Nclusters_max = 15
Nrepetitions = 100

qualities = []
inertias = []
models = []
kini = 1
kfin = 20
for k in range(kini,kfin+1):
    print("Evaluando k=%d" % k)
    km = KMeans(n_clusters=k,
                init='k-means++', n_init=Nrepetitions,
                max_iter=500, random_state=2)
    km.fit(X_km)
    models.append(km)
    inertias.append(km.inertia_)
    if k > 1:
        qualities.append(qmetric(X_km, km.labels_))
```

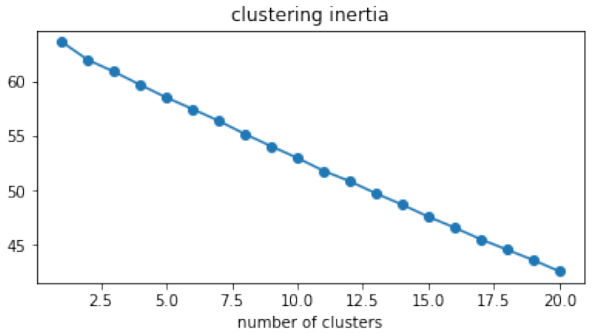
```
        #qualities.append(km.score(X_km))
    else:
        qualities.append(0)
```

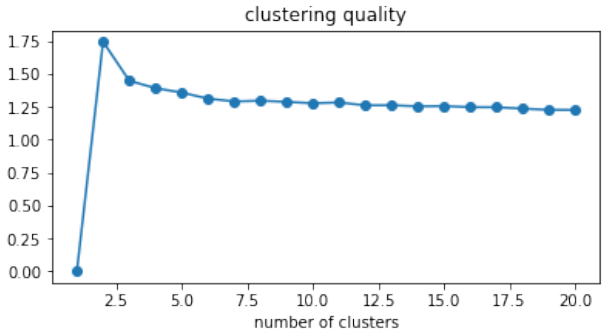
Output

```
(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 3.3.2 (python)

```python
fig = plt.figure(figsize=(14,3))

ax = plt.subplot(1,2,1)
plt.plot(range(kini,kfin+1), inertias, marker='o')
plt.xlabel('number of clusters')
plt.title('clustering inertia')

ax = plt.subplot(1,2,2)
plt.plot(range(kini,kfin+1), qualities, marker='o')
plt.xlabel('number of clusters')
plt.title('clustering quality')
plt.show()

best = pd.Series(qualities).idxmax() # get index for the best model
print("Best number of clusters", best)
km = models[best]
n_clusters = km.get_params()['n_clusters']
clusters = km.labels_
print ('Number of clusters of best quality', n_clusters)
```





Output

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

Script 3.3.3 (python)

```python
# We choose the best option to evaluate the quality of prediction
X = X_test
y = y_test
X_km = get_X_transform(X)
labels = km.fit_predict(X_km)
#print(labels)
# First we try with labels as is
labels_predicted = [str(label) for label in labels]
predicted = pd.Series(labels_predicted)
#print(labels_predicted)
print(metrics.classification_report(y, predicted))
print(metrics.confusion_matrix(y, predicted))

# Alternatively we invert the label to match the real labels of each group
labels_predicted = [str((label + 1)%2) for label in labels]
#print(labels_predicted)
predicted = pd.Series(labels_predicted)
print(metrics.classification_report(y, predicted))
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]]
```