NLP LAB 02 Théo Ripoll - Quentin Fish - Nicolas Fidel

Part 1: The Dataset

In [1]:
```
from datasets import load_dataset
from datasets import get_dataset_split_names
import pandas as pd
```

```
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/s
ite-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please upda
te jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/
user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
dataset = load_dataset("imdb")
dataset
```

```
Found cached dataset imdb (/Users/quentinfisch/.cache/huggingface/datasets/
imdb/plain_text/1.0.0/d613c88cf8fa3bab83b4ded3713f1f74830d1100e171db75bbddb
80b3345c9c0)
100%|██████████| 3/3 [00:00<00:00, 116.53it/s]
```

Out[2]:
```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
        num_rows: 50000
    })
})
```

In [10]:
```
get_dataset_split_names("imdb")
```

Out[10]:
```
['train', 'test', 'unsupervised']
```

Let's count the number of labs in each dataset

In [11]:
```
train_labels = pd.DataFrame(dataset["train"]['label'], columns=["label"])
print(train_labels.groupby("label")["label"].count())

test_labels = pd.DataFrame(dataset["test"]['label'], columns=["label"])
print(test_labels.groupby("label")["label"].count())
```

```
label
0    12500
1    12500
Name: label, dtype: int64
label
0    12500
1    12500
Name: label, dtype: int64
```

Question 1: How many splits does the dataset has?

There are 3 splits: `train`, `test` and `unsupervised`

Question 2: How big are the splits ?

train: 25000 test: 25000 unsupervised: 50000

Question 3: What is the proportion of each class on the supervised splits?

train: 50% positive, 50% negative test: 50% positive, 50% negative

Partie 2: Naive Bayes classifier

In [3]:

```
from string import punctuation
import re

def preprocess(dataset: pd.DataFrame) -> pd.DataFrame :
    """
    Preprocess the dataset by lowercasing the text and removing the punctuat

    Parameters
    ----------
    dataset : pd.DataFrame
        The dataset to preprocess

    Returns
    -------
    pd.DataFrame
        The preprocessed dataset
    """
    # First lower the case
    dataset["document"] = dataset["document"].apply(lambda x: x.lower())
    # Replace the punctuation with spaces. We keep the ' - that may give rev
    # Replace HTML tag <br />
    punctuation_to_remove = '|'.join(map(re.escape, sorted(list(filter(lambd
    print(f"Deleting all these punctuation: {punctuation_to_remove}")
    dataset["document"] = dataset["document"].apply(lambda x: re.sub(punctua
    return dataset
```

Apply the preprocessing steps to both the training and test sets. We choose to save them in a pandas DataFrame.

In [4]:
```
train_raw = pd.DataFrame(dataset["train"], columns=["text", "label"]).rename
preprocessed_train = preprocess(train_raw)
preprocessed_train
```

```
Deleting all these punctuation: \~|\}|\||\{|`|_|\^|\]|\\|\[|@|\?|>|=|<|;|:
|/|\.|,|\+|\*|\)|\(|\&|%|\$|\#|"|!
```

Out[4]:

	document	class
0	i rented i am curious-yellow from my video sto...	0
1	i am curious yellow is a risible and preten...	0
2	if only to avoid making this type of film in t...	0
3	this film was probably inspired by godard's ma...	0
4	oh brother after hearing about this ridicul...	0
...	...	...
24995	a hit at the time but now better categorised a...	1
24996	i love this movie like no other another time ...	1
24997	this film and it's sequel barry mckenzie holds...	1
24998	'the adventures of barry mckenzie' started lif...	1
24999	the story centers around barry mckenzie who mu...	1

25000 rows × 2 columns

In [5]:
```
test_raw = pd.DataFrame(dataset["test"], columns=["text", "label"]).rename(c
preprocessed_test = preprocess(test_raw)
preprocessed_test
```

```
Deleting all these punctuation: \~|\}|\||\{|`|_|\^|\]|\\|\[|@|\?|>|=|<|;|:
|/|\.|,|\+|\*|\)|\(|\&|%|\$|\#|"|!
```

Out[5]:

	document	class
0	i love sci-fi and am willing to put up with a ...	0
1	worth the entertainment value of a rental esp...	0
2	its a totally average film with a few semi-alr...	0
3	star rating saturday night friday ...	0
4	first off let me say if you haven't enjoyed a...	0
...	...	...
24995	just got around to seeing monster man yesterda...	1
24996	i got this as part of a competition prize i w...	1
24997	i got monster man in a box set of three films ...	1
24998	five minutes in i started to feel how naff th...	1
24999	i caught this movie on the sci-fi channel rece...	1

25000 rows × 2 columns

Question 2: Naive Bayes Classifier using pseudo-code

In [6]:

```
import numpy as np
from typing import List
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.feature_extraction.text import CountVectorizer

def get_vocabulary(d: pd.DataFrame) -> List[str]:
    """
    Return the vocabulary of the dataset

    Parameters
    ----------
    d : pd.DataFrame

    Returns
    -------
    List[str]
        The vocabulary
    """
    res = list(set(" ".join(d["document"]).split(" ")))
    # Remove empty string and words without any letter
    res = list(filter(lambda x: x != "" and re.search("[a-zA-Z]", x), res))
    return res

def train_naive_bayes(d: pd.DataFrame):
    """
    Train a Naive Bayes classifier
    Apply pseudo code from lecture 2

    Parameters
    ----------
```

```python
    d : pd.DataFrame

    Returns
    -------
    logprior : dict
        The log prior of each class
    loglikelihood : dict
        The log likelihood of each word for each class
    V : List[str]
        The vocabulary
    """
    classes = d["class"].unique()
    logprior = {}
    bigdoc = {}
    loglikelihood = {}
    V = get_vocabulary(d)
    for c in classes:
        count = {}
        n_doc = len(d)
        n_c = len(d[d["class"] == c])
        logprior[c] = np.log(n_c / n_doc)
        bigdoc[c] = list(" ".join(d[d["class"] == c]["document"]).split(" "))
        for word in V:
            count[(word, c)] = bigdoc[c].count(word)
        for word in V:
            loglikelihood[(word, c)] = np.log((count[(word, c)] + 1) / (sum(
    return logprior, loglikelihood, V

def test_naive_bayes(testdoc, classes, logprior, loglikelihood, V) -> int:
    """
    Test a Naive Bayes classifier

    Parameters
    ----------
    testdoc : str
        The document to classify
    classes : List[int]
        The list of classes
    logprior : dict
        The log prior of each class
    loglikelihood : dict
        The log likelihood of each word for each class
    V : List[str]
        The vocabulary

    Returns
    -------
    int
        The predicted class
    """
    sum_loglikelihood = {}
    for c in classes:
        sum_loglikelihood[c] = logprior[c]
        for word in testdoc.split(" "):
            if word in V:
```

```
                sum_loglikelihood[c] += loglikelihood[(word, c)]
    return max(sum_loglikelihood, key=sum_loglikelihood.get)
```

In [22]:
```
logprior_r, loglikelyhood_r, V_r = train_naive_bayes(preprocessed_train)

all_res = []
for row in preprocessed_test.iterrows():
    test_doc = row[1]["document"]
    res = test_naive_bayes(test_doc, preprocessed_test["class"].unique(), lo
    all_res.append(res)

print("Manual Naive Bayes Accuracy Score -> ",accuracy_score(preprocessed_te
print("Manual Naive Bayes Precision Score -> ",precision_score(preprocessed_
print("Manual Naive Bayes Recall Score -> ",recall_score(preprocessed_test["
```

```
Manual Naive Bayes Accuracy Score ->  81.364
Manual Naive Bayes Precision Score ->  85.78077941042255
Manual Naive Bayes Recall Score ->  75.19200000000001
```

Question 3: Naive Bayes Classifier using sklearn (Pipeline with CountVectorizer and MultinomialNB)

We will create a pipeline with a CountVectorizer and a MultinomialNB. We will use the default parameters for both of them as a first try.

In [7]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
```

In [8]:
```
def sklearn_naive_bayes(d_train: pd.DataFrame, pipeline_params: dict = {}) -
    """
    Train a Naive Bayes classifier using sklearn

    Parameters
    ----------
    d_train : pd.DataFrame
        The training dataset
    pipeline_params : dict, optional
        The parameters of the pipeline, by default {}

    Returns
    -------
    Pipeline
        The trained pipeline
    """
    # create pipeline
    pipeline = Pipeline([
        ('vectorizer', CountVectorizer()),
        ('classifier', MultinomialNB())
    ])
    pipeline.set_params(**pipeline_params)

    # train the model
    pipeline.fit(d_train["document"], d_train["class"])
    return pipeline
```

```
def test_sklearn_naive_bayes(pipeline: Pipeline, d_test: pd.DataFrame) -> Li
    """
    Test a Naive Bayes classifier using sklearn

    Parameters
    ----------
    pipeline : Pipeline
        The trained pipeline
    d_test : pd.DataFrame
        The test dataset

    Returns
    -------
    List[int]
        The predicted classes
    """
    # predict the labels on validation dataset
    predictions = pipeline.predict(d_test["document"])

    print("Sklearn Naive Bayes Accuracy Score -> ",accuracy_score(d_test["cl
    print("Sklearn Naive Bayes Precision Score -> ",precision_score(d_test["
    print("Sklearn Naive Bayes Recall Score -> ",recall_score(d_test["class"

    return predictions
```

In [9]:

```
pipeline = sklearn_naive_bayes(preprocessed_train)
predictions = test_sklearn_naive_bayes(pipeline, preprocessed_test)
```

```
Sklearn Naive Bayes Accuracy Score ->  81.44
Sklearn Naive Bayes Precision Score ->  86.05504587155963
Sklearn Naive Bayes Recall Score ->  75.03999999999999
```

Question 4: Report the accuracy on the test set

See prints above

Question 5: Most likely, the scikit-learn implementation will give better results. Looking at the documentation, explain why it could be the case.

The scikit-learn implementation is better because it uses a MultinomialNB which is a more efficient way to compute the probabilities. It also uses a CountVectorizer which is a more efficient way to count the words in the dataset.

Question 6: Why is accuracy a sufficient measure of evaluation here?

Because the dataset is balanced, we have the same number of positive and negative reviews. So the accuracy is a good measure of evaluation.

Question 7: Using one of the implementation, take at least 2 wrongly classified example from the test set and try explaining why the model failed.

In [21]:
```
# We will take a look at the sklearn implementation
# First we need to get the wrongly classified examples
wrongly_classified = preprocessed_test[preprocessed_test["class"] != predict

# We will take the first 2 examples
# We can see that the first example is a negative review but the model predi
# The second example is a positive review but the model predicted it as a ne
print(wrongly_classified.iloc[0]["document"])
print(wrongly_classified.iloc[1]["document"])
print()

# Let's see the probability of each class for the first example
print(pipeline.predict_proba([wrongly_classified.iloc[0]["document"]]))
# Let's see the probability of each class for the second example
print(pipeline.predict_proba([wrongly_classified.iloc[1]["document"]]))
```

```
blind date  columbia pictures  1934   was a decent film  but i have a few i
ssues with this film  first of all  i don't fault the actors in this film a
t all  but more or less  i have a problem with the script  also  i understa
nd that this film was made in the 1930's and people were looking to escape
reality  but the script made ann sothern's character look weak  she kept go
ing back and forth between suitors and i felt as though she should have sta
yed with paul kelly's character in the end  he truly did care about her and
her family and would have done anything for her and he did by giving her up
in the end to fickle neil hamilton who in my opinion was only out for a goo
d time  paul kelly's character  although a workaholic was a man of integrit
y and truly loved kitty  ann sothern  as opposed to neil hamilton  while he
did like her a lot  i didn't see the depth of love that he had for her char
acter  the production values were great  but the script could have used a l
ittle work
ben   rupert grint   is a deeply unhappy adolescent  the son of his unhappi
ly married parents  his father   nicholas farrell   is a vicar and his moth
er   laura linney   is     well  let's just say she's a somewhat hypocritic
al soldier in jesus' army  it's only when he takes a summer job as an assis
tant to a foul-mouthed  eccentric  once-famous and now-forgotten actress ev
ie walton   julie walters   that he finally finds himself in true 'harold a
nd maude' fashion  of course  evie is deeply unhappy herself and it's only
when these two sad sacks find each other that they can put their mutual mis
ery aside and hit the road to happiness of course it's corny and sentimenta
l and very predictable but it has a hard side to it  too and walters  who c
ould sleep-walk her way through this sort of thing if she wanted  is excell
ent  it's when she puts the craziness to one side and finds the pathos in t
he character   like hitting the bottle and throwing up in the sink   that s
he's at her best  the problem is she's the only interesting character in th
e film  and it's not because of the script which doesn't do anybody any fav
ours   grint  on the other hand  isn't just unhappy  he's a bit of a bore a
s well while linney's starched bitch is completely one-dimensional   still
she's got the english accent off pat   the best that can be said for it is
that it's mildly enjoyable - with the emphasis on the mildly

[[4.22158007e-06 9.99995778e-01]]
[[0.00150068 0.99849932]]
```

We can see that the model is very confident about its prediction for the two examples (0.99...) but it's wrong. These examples are very hard to classify because they are very close to the decision boundary and also mixing a movie description (which can have positive or negative connotations due to the life of the main character, etc) and a review. So the model is not able to classify them correctly because of the confusing bundary between description and facts and the opinion.

Question 8: What are the top 10 most important words (features) for each class? (bonus points)

In [10]:

```python
# We will use the sklearn implementation to get the top 10 most important wo

def get_top_10_words(pipeline: Pipeline) -> dict:
    """
    Get the top 10 words for each class
```

```
    Parameters
    ----------
    pipeline : Pipeline
        The trained pipeline

    Returns
    -------
    dict
        The top 10 words for each class
    """
    top_10_words = {}
    for c in preprocessed_test["class"].unique():
        loglikelihood = pipeline.named_steps["classifier"].feature_log_prob_
        V = pipeline.named_steps["vectorizer"].vocabulary_
        top_10_words[c] = [list(V.keys())[list(V.values()).index(i)] for i i
    return top_10_words
```

In [11]:

```
get_top_10_words(pipeline)
```

Out[11]:

```
{0: ['was', 'that', 'this', 'in', 'it', 'is', 'to', 'of', 'and', 'the'],
 1: ['as', 'this', 'that', 'it', 'in', 'is', 'to', 'of', 'and', 'the']}
```

The words we retreive are stop words, so they are not very meaningful. Let's try to remove them and see if we get better results.

In [24]:

```
pipeline_without_stopwords = sklearn_naive_bayes(preprocessed_train, {"vecto
predictions_without_stopwords = test_sklearn_naive_bayes(pipeline_without_st

get_top_10_words(pipeline_without_stopwords)
```

```
Sklearn Naive Bayes Accuracy Score ->  81.976
Sklearn Naive Bayes Precision Score ->  86.22439731738264
Sklearn Naive Bayes Recall Score ->  76.112
```

Out[24]:

```
{0: ['story',
  'don',
  'time',
  'really',
  'bad',
  'good',
  'just',
  'like',
  'film',
  'movie'],
 1: ['people',
  'really',
  'great',
  'time',
  'story',
  'just',
  'good',
  'like',
  'movie',
  'film']}
```

We see that the top 10 words are more unique using stopwords, but the results are pretty equivalent with or without stopwords.

Question 9: Play with scikit-learn's version parameters. For example, see if you can consider unigram and bigram instead of only unigrams.

We will compare previous results using sklearn with the results using unigram and bigram, and with/without removing stopwords.

In [25]:
```
# Unigram and bigram
pipeline_bigram = sklearn_naive_bayes(preprocessed_train, {"vectorizer__ngra
predictions_bigram = test_sklearn_naive_bayes(pipeline_bigram, preprocessed_
```

```
Sklearn Naive Bayes Accuracy Score ->  84.244
Sklearn Naive Bayes Precision Score ->  87.4857693318154
Sklearn Naive Bayes Recall Score ->  79.92
```

In [26]:
```
# Unigram and bigram with stopwords
pipeline_bigram_stopwords = sklearn_naive_bayes(preprocessed_train, {"vector
predictions_bigram_stopwords = test_sklearn_naive_bayes(pipeline_bigram_stop
```

```
Sklearn Naive Bayes Accuracy Score ->  85.672
Sklearn Naive Bayes Precision Score ->  88.62612612612612
Sklearn Naive Bayes Recall Score ->  81.848
```

In [27]:
```
# Only bigram
pipeline_only_bigram = sklearn_naive_bayes(preprocessed_train, {"vectorizer_
predictions_only_bigram = test_sklearn_naive_bayes(pipeline_only_bigram, pre
```

```
Sklearn Naive Bayes Accuracy Score ->  82.952
Sklearn Naive Bayes Precision Score ->  87.63018454229857
Sklearn Naive Bayes Recall Score ->  76.736
```

In [28]:
```
# Only bigram with stopwords
pipeline_only_bigram_stopwords = sklearn_naive_bayes(preprocessed_train, {"v
predictions_only_bigram_stopwords = test_sklearn_naive_bayes(pipeline_only_b
```

```
Sklearn Naive Bayes Accuracy Score ->  86.952
Sklearn Naive Bayes Precision Score ->  89.35753237900477
Sklearn Naive Bayes Recall Score ->  83.896
```

The accuracy is better with only bigrams and without removing stopwords.

Part 3: Stemming & Lemmatization

In this part we will add preprocessing, including stemming and leammatization.

We need to add an extra module for spacy.

In [ ]:
```
! python -m spacy download en_core_web_sm
```

Lemmatization preprocessing

Let's start with a small example to understand how to recover a lem.

In this case we will use Spacy, especially its pipeline features to do preprocessing.

In [14]:
```python
# Setup spacy
import spacy
nlp = spacy.load('en_core_web_sm')
```

In [30]:
```python
# Take a 20 characters sentence example from the test dataset
test_list = dataset['train']['text'][0].split()[:20]
test_sentence = ' '.join(test_list)

# Lemmatize the sentence
doc = nlp(test_sentence)

# Get all token
tokens = [token.text for token in doc]

print(f'Original Sentence: {test_sentence}')
for token in doc:
    if token.text != token.lemma_:
        print(f'Original : {token.text}, New: {token.lemma_}')
```

```
Original Sentence: I rented I AM CURIOUS-YELLOW from my video store because
of all the controversy that surrounded it when it was
Original : rented, New: rent
Original : AM, New: be
Original : CURIOUS, New: curious
Original : surrounded, New: surround
Original : was, New: be
```

Results look good, words are reduced to their root form.

Let's define a preprocessing function.

In [15]:
```python
def lemma_preprocessor(x_list: List[str]) -> List[str]:
    """
    Preprocessing function to lowercase and remove punctuation
    of a list of string and lemmatize each string.

    Args:
        x_list: List of strings

    Returns:
        List of preprocessed strings.
    """
    no_punc_lower = [x.lower().translate(str.maketrans("", "", punctuation))
    spacy_nlp = spacy.load('en_core_web_sm')
    res = []
    for sentence in no_punc_lower:
        doc = spacy_nlp(sentence)
        s = []
        for word in doc:
            s.append(word.lemma_)
        s = ' '.join(s)
```

```
        res.append(s)
    return res
```

Print a example of the result :

In [26]:
```
print(dataset['train']['text'][:1])
lemma_preprocessor(dataset['train']['text'][:1])
```

```
['I rented I AM CURIOUS-YELLOW from my video store because of all the contr
oversy that surrounded it when it was first released in 1967. I also heard
that at first it was seized by U.S. customs if it ever tried to enter this
country, therefore being a fan of films considered "controversial" I really
had to see this for myself.<br /><br />The plot is centered around a young
Swedish drama student named Lena who wants to learn everything she can abou
t life. In particular she wants to focus her attentions to making some sort
of documentary on what the average Swede thought about certain political is
sues such as the Vietnam War and race issues in the United States. In betwe
en asking politicians and ordinary denizens of Stockholm about their opinio
ns on politics, she has sex with her drama teacher, classmates, and married
men.<br /><br />What kills me about I AM CURIOUS-YELLOW is that 40 years ag
o, this was considered pornographic. Really, the sex and nudity scenes are
few and far between, even then it\'s not shot like some cheaply made porno.
While my countrymen mind find it shocking, in reality sex and nudity are a
major staple in Swedish cinema. Even Ingmar Bergman, arguably their answer
to good old boy John Ford, had sex scenes in his films.<br /><br />I do com
mend the filmmakers for the fact that any sex shown in the film is shown fo
r artistic purposes rather than just to shock people and make money to be s
hown in pornographic theaters in America. I AM CURIOUS-YELLOW is a good fil
m for anyone wanting to study the meat and potatoes (no pun intended) of Sw
edish cinema. But really, this film doesn\'t have much of a plot.']
```

Out[26]:
```
['I rent I be curiousyellow from my video store because of all the controve
rsy that surround it when it be first release in 1967 I also hear that at f
irst it be seize by us customs if it ever try to enter this country therefo
re be a fan of film consider controversial I really have to see this for my
selfbr br the plot be center around a young swedish drama student name lena
who want to learn everything she can about life in particular she want to f
ocus her attention to make some sort of documentary on what the average swe
de think about certain political issue such as the vietnam war and race iss
ue in the united states in between ask politician and ordinary denizen of s
tockholm about their opinion on politic she have sex with her drama teacher
classmate and married menbr br what kill I about I be curiousyellow be that
40 year ago this be consider pornographic really the sex and nudity scene b
e few and far between even then its not shoot like some cheaply make porno
while my countryman mind find it shock in reality sex and nudity be a major
staple in swedish cinema even ingmar bergman arguably their answer to good
old boy john ford have sex scene in his filmsbr br I do commend the filmmak
er for the fact that any sex show in the film be show for artistic purpose
rather than just to shock people and make money to be show in pornographic
theater in america I be curiousyellow be a good film for anyone want to stu
dy the meat and potatoe no pun intend of swedish cinema but really this fil
m do not have much of a plot']
```

We see that the preprocessing is working well: words are reduced to their lemma.

Stemming preprocessing

Let's start with a small example to understand how to recover a lem.

In this case we will use NLTK, another library than Spacy, but it offers stemming unlike Spacy

In [17]:
```
import nltk

from nltk.stem import PorterStemmer
nltk.download("punkt")
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/quentinfisch/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[17]: True

In [18]:
```
# Initialize Python porter stemmer
ps = PorterStemmer()

test_list = dataset['train']['text'][0].split()[:20]
test_sentence = ' '.join(test_list)

# Example inflections to reduce
example_words = ["program","programming","programer","programs","programmed"

print(f'Original Sentence: {test_sentence}')
# Perform stemming
print("{0:20}{1:20}".format("--Word--","--Stem--"))
for word in test_list:
    print ("{0:20}{1:20}".format(word, ps.stem(word)))
```

```
Original Sentence: I rented I AM CURIOUS-YELLOW from my video store because
of all the controversy that surrounded it when it was
--Word--            --Stem--
I                   i
rented              rent
I                   i
AM                  am
CURIOUS-YELLOW      curious-yellow
from                from
my                  my
video               video
store               store
because             becaus
of                  of
all                 all
the                 the
controversy         controversi
that                that
surrounded          surround
it                  it
when                when
it                  it
was                 wa
```

Again, results are stasisfyng. However, we observe some errors, such as "becaus" instead of "because", or "wa" instead of "was".

Let's define a preprocessing function.

In [19]:
```
def stem_preprocessor(x_list: List[str]) -> List[str]:
    """
    Preprocessing function to stem each string.

    Args:
        x_list: List of strings

    Returns:
        List of preprocessed strings.
    """
    spacy_nlp = spacy.load('en_core_web_sm')
    res = []
    ps = PorterStemmer()
    for sentence in x_list:
        doc = spacy_nlp(sentence)
        s = []
        for word in doc:
            s.append(ps.stem(str(word)))
        s = ' '.join(s)
        res.append(s)
    return res
```

In [36]:
```
example_words = ["program","programming","programer","programs","programmed"
stem_preprocessor(example_words)
```

Out[36]:
```
['program', 'program', 'program', 'program', 'program']
```

Training with Stem and Lemmatize

Lemma training

Both are working well. Now let's try to use lemmatization in our pipeline

In [27]:
```
# use stem_preprocessor to preprocess the training and test data
preprocessed_train_lemma = lemma_preprocessor(train_raw["document"][:2])
preprocessed_train_lemma
```

Out[27]:
```
['I rent I be curiousyellow from my video store because of all the controve
rsy that surround it when it be first release in 1967   I also hear that at
first it be seize by u s   custom if it ever try to enter this country   th
erefore be a fan of film consider   controversial   I really have to see th
is for myself the plot be center around a young swedish drama student name
lena who want to learn everything she can about life   in particular she wa
nt to focus her attention to make some sort of documentary on what the aver
age swede think about certain political issue such as the vietnam war and r
ace issue in the united states   in between ask politician and ordinary den
izen of stockholm about their opinion on politic   she have sex with her dr
ama teacher   classmate   and married man what kill I about I be curiousyel
low be that 40 year ago   this be consider pornographic   really   the sex
and nudity scene be few and far between   even then its not shoot like some
cheaply make porno   while my countryman mind find it shocking   in reality
sex and nudity be a major staple in swedish cinema   even ingmar bergman
arguably their answer to good old boy john ford   have sex scene in his fil
m I do commend the filmmaker for the fact that any sex show in the film be
show for artistic purpose rather than just to shock people and make money t
o be show in pornographic theater in america   I be curiousyellow be a good
film for anyone want to study the meat and potato   no pun intend   of swed
ish cinema   but really   this film do not have much of a plot',
 '  I be curious   yellow   be a risible and pretentious steaming pile   it
do not matter what one political view be because this film can hardly be ta
ke seriously on any level   as for the claim that frontal male nudity be an
automatic nc17   that be not true   I ve see rrate film with male nudity
grant   they only offer some fleeting view   but where be the rrate film wi
th gape vulvas and flap labia   nowhere   because they do not exist   the s
ame go for those crappy cable show   schlong swinge in the breeze but not a
clitoris in sight   and those pretentious indie movie like the brown bunny
  in which be treat to the site of vincent gallos throb johnson   but not a
trace of pink visible on chloe sevigny   before cry   or imply   doublesta
ndard   in matter of nudity   the mentally obtuse should take into account
one unavoidably obvious anatomical difference between man and woman   there
be no genital on display when actress appear nude   and the same can not be
say for a man   in fact   you generally will not see female genital in an a
merican film in anything short of porn or explicit erotica   this allege do
ublestandard be less a double standard than an admittedly depressing abilit
y to come to term culturally with the inside of women body']
```

Now let's define a function that will drive the model by adding the preprocessor lemma to the pipeline

In [20]:
```python
from sklearn.preprocessing import FunctionTransformer

def sklearn_naive_bayes_lemma(d_train: pd.DataFrame, pipeline_params: dict =
    """
    Train a Naive Bayes classifier using sklearn with lemmatization.

    Parameters
    ----------
    d_train : pd.DataFrame
        The training dataset
    pipeline_params : dict, optional
        The parameters of the pipeline, by default {}
```

```
    Returns
    --------
    Pipeline
        The trained pipeline
    """
    # create pipeline with lemmatization, vectorizer and classifier
    pipeline = Pipeline([
        ('lemmatizer', FunctionTransformer(lemma_preprocessor)),
        ('vectorizer', CountVectorizer()),
        ('classifier', MultinomialNB())
    ])
    pipeline.set_params(**pipeline_params)

    # train the model
    pipeline.fit(d_train["document"], d_train["class"])
    return pipeline
```

Training and evaluation of the model again with these pretreatment :

In [23]:
```
pipeline_lemma = sklearn_naive_bayes_lemma(train_raw)
predictions_lemma = test_sklearn_naive_bayes(pipeline_lemma, test_raw)
```

```
Sklearn Naive Bayes Accuracy Score ->  80.976
Sklearn Naive Bayes Precision Score ->  85.6078719882288
Sklearn Naive Bayes Recall Score ->  74.47200000000001
```

Results are not better than before (with default settings): 80.97% vs 81.44% accuracy. This is probably due to the fact that the lemmatization is not very efficient in this case. This can be caused by the fact the language is English, and the lemmatization is not very efficient for this language because of it's low morphology, removing information that could be useful for the classifier.

Let's try with stemming.

Stem training

Now let's define a function that will drive the model by adding the preprocessor stem to the pipeline

In [21]:
```
from sklearn.preprocessing import FunctionTransformer

def sklearn_naive_bayes_stem(d_train: pd.DataFrame, pipeline_params: dict =
    """
    Train a Naive Bayes classifier using sklearn with lemmatization.

    Parameters
    ----------
    d_train : pd.DataFrame
        The training dataset
    pipeline_params : dict, optional
        The parameters of the pipeline, by default {}

    Returns
```

```
    -------
    Pipeline
        The trained pipeline
    """
    # create pipeline with lemmatization, vectorizer and classifier
    pipeline = Pipeline([
        ('lemmatizer', FunctionTransformer(stem_preprocessor)),
        ('vectorizer', CountVectorizer()),
        ('classifier', MultinomialNB())
    ])
    pipeline.set_params(**pipeline_params)

    # train the model
    pipeline.fit(d_train["document"], d_train["class"])
    return pipeline
```

In [24]:
```
pipeline_stem = sklearn_naive_bayes_stem(train_raw)
predictions_stem = test_sklearn_naive_bayes(pipeline_stem, test_raw)
```

```
Sklearn Naive Bayes Accuracy Score ->  80.696
Sklearn Naive Bayes Precision Score ->  85.29898804047839
Sklearn Naive Bayes Recall Score ->  74.176
```

Here the results are even worse than before (with default settings): 80.69% vs 81.44% accuracy. Again, we surely have the same problem as before, the stemming is not very efficient in this case. Lemmatization is better than stemming in this case, because it's more aggressive on words changed.