Introduction to Natural Language Processing Catch-up 1

Authors :

- Tony George

Introduction

(Copied from subject)

In this small project, you will code a sentiment classifier using the naive Bayes methods seen in class and compare it with the FastText library. There are a few theoritical questions to answer as well.

Please, read the full assignment before starting.

For coding standards, please respect the following guidelines

- Use docstring format to describe your functions and their arguments.
- Use typing.
- Have clear and verbatim variable names (not x, x1, x2, xx, another_x, ...).
- Make your results reproducible (force random seeds values when necessary and possible).
- Don't hesitate commenting in details part of the code you consider complex or hard to read.

Do not hesitate contacting me if you have any question, but please don't wait until the last moment to do so.

Imports

Using conda with python version 3.8. A conda yml should be available if I didn't forget to generate it.

In [1]:
```
# TODO : Generate conda yml
import numpy as np
import matplotlib.pyplot as plt
```

Forcing seed

This helps reproducibility, feel free to play with !

In [2]:
```
from typing import Dict, List, Tuple
np.random.seed(42)
```

The dataset

Using HuggingFace's version of the IMDB dataset as asked by subject.

Importing from HuggingFace

In [3]:
```
from datasets import load_dataset, load_dataset_builder

ds_info = load_dataset_builder("imdb")
print("Desc :", ds_info.info.description)
print("Features :", ds_info.info.features)
print("Splits :", ds_info.info.splits)
```

```
Desc : Large Movie Review Dataset.
This is a dataset for binary sentiment classification containing substantially more data
than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews
for training, and 25,000 for testing. There is additional unlabeled data for use as wel
l.
Features : {'text': Value(dtype='string', id=None), 'label': ClassLabel(num_classes=2, n
ames=['neg', 'pos'], names_file=None, id=None)}
Splits : {'train': SplitInfo(name='train', num_bytes=33432835, num_examples=25000, datas
et_name='imdb'), 'test': SplitInfo(name='test', num_bytes=32650697, num_examples=25000,
dataset_name='imdb'), 'unsupervised': SplitInfo(name='unsupervised', num_bytes=67106814,
num_examples=50000, dataset_name='imdb')}
```

In [4]:
```
# Cheating a bit by downloading the DS now
ds_train    = load_dataset("imdb", split = "train")
ds_test     = load_dataset("imdb", split = "test")
```

```
Reusing dataset imdb (C:\Users\Griffures\.cache\huggingface\datasets\imdb\plain_text\1.
0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f0561f9a)
Reusing dataset imdb (C:\Users\Griffures\.cache\huggingface\datasets\imdb\plain_text\1.
0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f0561f9a)
```

Dataset Exploration

How many splits does the dataset has?

In [5]:
```
# Using the builder to get info on splits.
ds_info.info.splits.keys()
```

Out[5]:
```
dict_keys(['train', 'test', 'unsupervised'])
```

HuggingFaces's version of the IMDB dataset has 3 different splits, though we will only interest ourselves in the first two ones.

The *train* and *test* split are your standard ML splits, while the *unsupervised* data contains unlabelled data for those willing to gain more volumetry at the cost of some work.

How big are these splits?

In [6]:
```
# This small lib is great to convert hard numbers into a human-readable format.
from humanize import naturalsize
from datasets import SplitInfo

def split_info_on(sp_info : SplitInfo) -> str:
    """
    :param sp_info: The SplitInfo object from HuggingFace's datasets lib.
    :return: Human readable string with a quick description on the object.
    """
    return f"""Split {sp_info.name} contains {sp_info.num_examples} examples ({naturalsi

# Coming from Scala, having to wrap a map object in Python makes me sad.
# You will further down that my mindset is heavily ~~~corrupted~~~ inspired by MapReduce
list(map(split_info_on, ds_info.info.splits.values()))
```

Out[6]:
```
['Split train contains 25000 examples (33.4 MB).',
 'Split test contains 25000 examples (32.7 MB).',
```

```
 'Split unsupervised contains 50000 examples (67.1 MB).']
```

In term of size, we have a 50/50 split between labelled and unlabelled data, and another 50/50 between the *train* and *test* split for the former one.

What is the proportion of each class on the supervised splits?

In [7]:
```
from datasets import Dataset
from typing import Dict

def class_histogram(ds : Dataset) -> Dict[str, int]:
    """
    :param ds: Dataset object to dress the class histogram of.
    :return: List of Tuple2 [('class_label', count), ...]
    """
    labels = ds.info.features['label'].names
    # Extracting the label from the example, then counting occurrences using np.unique()
    _, counts = np.unique(ds['label'], return_counts = True)
    return dict(zip(labels, counts))
```

In [8]:
```
print("Train :", class_histogram(ds_train))
print("Test :", class_histogram(ds_test))
```

```
Train : {'neg': 12500, 'pos': 12500}
Test : {'neg': 12500, 'pos': 12500}
```

Both our splits have no bias toward a class or the other, with again a perfect 50/50 split.

Naive Bayes classifier

Pretreatment

Using the standard NLP pretreatment workflow, up to stemming.

The reason I don't do lemmatization is because my pretreatment function is simple and tends to butcher words. (i.e. "you'll" becomes ["you", "'ll"], and wordnet lemmatizer does not recognize the second word as will).

In [9]:
```
# Downloading NLTK (we will use it below anyway).
import nltk
nltk.download('punkt')
nltk.download('stopwords')

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from string import punctuation
from nltk.stem import LancasterStemmer

stops_en = set(stopwords.words('english'))
lancaster = LancasterStemmer()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Griffures\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Griffures\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [10]:
```
def pretreatment(text : str) -> str:
    # Lowercasing
```

```
    raw_words = word_tokenize(text.lower())

    # Filter out : punctuation & stop stopwords
    is_relevant = lambda word : not (all(map(lambda c: c in punctuation, list(word))) #
                                     or word in stops_en) # BONUS : Removing stopwords
    filtered_words = list(filter(is_relevant, raw_words))

    # BONUS : Stemming
    lemmatized_words = list(map(lancaster.stem, filtered_words))

    return  ' '.join(lemmatized_words)
```

In [11]:
```
# Select an example here
ex_preprocessing = ds_test[0]['text']

print("Example with :", ex_preprocessing)
print("Which gives :", pretreatment(ex_preprocessing))
```

```
Example with : I went and saw this movie last night after being coaxed to by a few frien
ds of mine. I'll admit that I was reluctant to see it because from what I knew of Ashton
Kutcher he was only able to do comedy. I was wrong. Kutcher played the character of Jake
Fischer very well, and Kevin Costner played Ben Randall with such professionalism. The s
ign of a good movie is that it can toy with our emotions. This one did exactly that. The
entire theater (which was sold out) was overcome by laughter during the first half of th
e movie, and were moved to tears during the second half. While exiting the theater I not
only saw many women in tears, but many full grown men as well, trying desperately not to
let anyone see them crying. This movie was great, and I suggest that you go see it befor
e you judge.
Which gives : went saw movy last night coax friend min 'll admit reluct see knew ashton
kutch abl comedy wrong kutch play charact jak fisch wel kevin costn play ben randal prof
ess sign good movy toy emot on exact entir the sold overcom laught first half movy mov t
ear second half exit the saw many wom tear many ful grown men wel try desp let anyon see
cry movy gre suggest go see judg
```

Implementing the model

As to not rely too much on sklearn's library, and only using it to gain time, we will override the preprocessing function of the CountVectorizer with our own, and disable its stopwords list.

The model will actually take a hit from this workflow (both in performance, as the tokenizing will actually happen twice, and in accuracy, as we could do everything with a well configured CountVectorizer). However, it proves a greater comprehension of the notions at play.

In [12]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
import numpy.typing as npt
from typing import Tuple


class CustomNaiveBayes:
    """
    Wrapper class to articulate sklearn's CountVectorizer and MultinomialNB models.
    """
    vectorizer = CountVectorizer(preprocessor = pretreatment, stop_words = None)
    clf = MultinomialNB()
    labels = []

    def fit(self, ds : Dataset) -> None:
        """
        Wrapper function to launch a complete train workflow from a HF's dataset
        :param ds: HuggingFace DataSet object to train on.
        :return: Nothing.
        """
```

```
        # Extracting labels for predict_label function
        self.labels = ds.features['label'].names
        # Shuffling and extracting features
        ds.shuffle()
        X_as_docs, y_as_ints = self.DS_to_XnY(ds)
        X_as_features = self.vectorizer.fit_transform(X_as_docs)
        # Actual training
        self.clf.fit(X_as_features, y_as_ints)

    def predict(self, X_as_docs : npt.ArrayLike) -> npt.NDArray[int]:
        """
        Extract features then launch model's predictions on a list of documents
        :param X_as_docs: List of documents (corpus)
        :return: Predicted classes as ints
        """
        X_as_features = self.vectorizer.transform(X_as_docs)
        return self.clf.predict(X_as_features)

    def predict_label(self, X_as_docs : npt.ArrayLike) -> npt.NDArray[str]:
        """
        Same as predict, but return the classes label
        :param X_as_docs: List of documents (corpus)
        :return: Predicted classes as strings
        """
        predictions = self.predict(X_as_docs)
        return np.fromiter([self.labels[y] for y in predictions], str)

    def score(self, ds : Dataset) -> float:
        """
        Evaluate the model with a HF's dataset.
        :param ds: HuggingFace DataSet object to evaluate upon.
        :return: Accuracy as float (0~1).
        """
        X_as_docs, y_as_ints = self.DS_to_XnY(ds)
        X_as_features = self.vectorizer.transform(X_as_docs)
        return self.clf.score(X_as_features, y_as_ints)

    def DS_to_XnY(self, ds : Dataset) -> Tuple[npt.ArrayLike, npt.ArrayLike]:
        """
        Transforms a HuggingFace dataset object to a more usual X and y tuple for fittin
        :param ds: Dataset to convert
        :return: Tuple : X as list[str] and Y as list[int], both of the same len.
        """
        raw_corpus = np.array(ds['text'])
        targets = np.array(ds['label'])
        return raw_corpus, targets
```

Training

In [13]:
```
customNB = CustomNaiveBayes()
customNB.fit(ds_train)
```

```
Loading cached shuffled indices for dataset at C:\Users\Griffures\.cache\huggingface\dat
asets\imdb\plain_text\1.0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f056
1f9a\cache-9d17e850e5680b7a.arrow
```

Reporting accuracy

In [14]:
```
customNB_acc = customNB.score(ds_test)
print("Accuracy :", customNB_acc)
```

```
Accuracy : 0.81168
```

Is accuracy it sufficient here ?

Accuracy is a sufficient metric here for multiple reasons :

- Since we only have 2 classes, a confusion matrix looses its relevance (p versus 1 - p).
- Due to the nature of our classification, we do not care specifically for type 1 nor type 2 errors. Therefore focusing Precision or Recall is meaningless.
- The above, plus the fact that our classes are perfectly balanced, makes f1-score a very similar metric as accuracy (harmonic mean between precision and recall).

Top 10 important feature from each class

I already removed the stop words, so we already have the best results ! :)

In [15]:
```
def top_10_features_per_class(model: CustomNaiveBayes) -> Dict:
    """
    Return the top ten heaviest features for each class according to the model
    :param model: the model to extract weights from
    :return: Dictionary : Class Label -> Features (feature(stem) -> log_value)
    """
    # Using a custom dtype to preserve info on features' names.
    feature_log_t = [("feature", '<U50'), ("log_value", float)]
    # Just stacks the features names (stems) on top of the logs array
    feature_stack = lambda logs : np.array(list(zip(model.vectorizer.get_feature_names_o
    # Sort the array, take the 10 last value, then flip it to get our top 10
    sort_10_biggest = lambda arr : np.flip(np.sort(arr, order="log_value")[-10:])

    # Applying the functions in the correct order
    logs_with_features = [sort_10_biggest(feature_stack(logs)) for logs in model.clf.fea

    return dict(zip(model.labels, map(dict, logs_with_features)))
```

In [16]:
```
# Simple extraction of the above dictionary
for label, features in top_10_features_per_class(customNB).items():
    print(f"Top 10 features (stems) in {label.upper()} :")
    print('\n'.join([f"  * {feature:10}{log:2.3}" for feature, log in features.items()])
```

```
Top 10 features (stems) in NEG :
  * br        -3.41
  * movy      -4.04
  * film      -4.25
  * on        -4.78
  * act       -4.79
  * lik       -4.87
  * ev        -4.89
  * real      -5.09
  * mak       -5.27
  * bad       -5.3
Top 10 features (stems) in POS :
  * br        -3.51
  * film      -4.18
  * movy      -4.32
  * on        -4.76
  * act       -5.05
  * lik       -5.06
  * real      -5.09
  * ev        -5.2
  * tim       -5.31
  * good      -5.35
```

Analysis :

Prevalent stems common in both classes (8/10) :

- br
- movy
- film
- on
- act
- like
- real
- ev

Prevalent NEG stems :

- mak
- bad

Prevalent POS stems :

- time
- good

A great number of steams are common between our two classes. This makes me think that a greater notion of context should be given to better the model. Just giving the CountVectorizer 2-grams would help it makes the difference between 'bad' and 'not bad', 'good act' and 'bad act', etc.

However 81% for such a simple model is not bad at all already ! (n-grams would explode the size of our vocabulary).

Digging up errors

In [17]:
```
def get_errors(model : CustomNaiveBayes, ds : Dataset, n):
    X_as_docs, Y = model.DS_to_XnY(ds.shuffle()[n * 100:])
    y_pred = model.predict(X_as_docs)
    errors = X_as_docs[y_pred != Y]
    return errors[:n]
```

In [18]:
```
five_errors = get_errors(customNB, ds_test, 5)
```

```
Loading cached shuffled indices for dataset at C:\Users\Griffures\.cache\huggingface\datasets\imdb\plain_text\1.0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f0561f9a\cache-e3a051aef336c446.arrow
```

In [19]:
```
for i, error in enumerate(five_errors):
    print(f"{i + 1}. {error}")
```

```
1. (the description of the mood of the movie may be considered as a spoiler - because there is not much action in fact)<br /><br />Great one...<br /><br />Is it for my peculiar interest for the dystopias and utopias? Is it for the atmosphere of the movie. Or is there some more magic? If yes, it is for sure the utmost human one...<br /><br />This film is, no doubt, extremely artistic/artificial (depends on taste). I can imagine most of the people who hate to watch slow movies (and those of Tsai Ming Liang (who I didn't enjoy other times) are one of the slowest that I know), suffering during the movie. Yes, people are unable to slow down and to let time pass - and to watch it without feeling they waste it. One can take this piece as torture or as a therapy...<br /><br />The topic at the surface? The lack of communication - even if we live in rabbit cages - one next to each other - but not really together? People are tired, sick of something and unable to describe it - just don't want to meet, touch, talk, confront the others... like if they had
```

disappeared. The big block of flats looks void and the rain falling constantly evokes th e strange melancholy inside. And sometimes it must be something abnormal, unexpected, so me unwanted decay as a hole in the floor of concrete - that allows us to reach each othe r.

One of the possible ways to look at it is this: Don't survey the inner wor ld of the characters - consider the whole movie-space to be inside of yourself. And ask - why is it there? Where could these depressive states and moods come form? Is there a p lace for them, they don't have a right to be here? And search for the answers (if you ne ed them) among the walls and halls of the block - instead of inside hardly transparent m ind of a man.

The key to understand is not-to-understand - to let a movie bor row us - as a subject of study - inside itself - and at the end safely return us to our more colorful and "normal" looking reality.

Then, maybe, you will reach - lik e me - the feeling of real, possible, non-pathetic hope, that in core we are still human s... and this state of mind can help one much to live in this world.

2. The trailer to this film focused so much on the chain (of course, because it's so sen sational) that it missed most of the movie, which is about a developing, although rather simply drawn, relationship between Lazarus and Rae as they attempt to recover from their past pains with each other.

Of course, with the premise of a nymphomaniac in chains, it's no surprise that there's plenty of implied sex involved. However, at it's c ore, Black Snake Moan is a basic tale of redemption and the healing power of helping ano ther person along. Maybe it's just me though, but I think poor Lazarus should've had his story focused on more. He's a hurting man after his wife leaves him, but we never fully see how helping Rae resolve her past pains heals him too. It's just implied that it does --in essence, he plays the wizard that helps the young Rae overcome her curse, through a big ol' chain and some blues.

I like the story, but I wish it were a bit more even and didn't have to rely on the sensational. The side characters were fairly decent, if simple and I liked the music. The acting was good enough, although I can't be certain if the Rae character is fully believable. But that might just be my naivety.

 All in all, I liked the film, but I wasn't compelled by it. Maybe it's that I'm too crit ical, but the story seems a little too convenient to be fully believable and so, while i t all seemed very cool, I could never truly buy it. The chain thing was a little too far -fetched for me. Still, this can provide some entertainment for those looking for dramat ic redemption stories with a shot of the blues. 7/10

3. I haven't written a review on here in ages but rewatching all of bottom TV show, live shows and this I felt I had to make my views on this movie known! It is, I feel, the per fect comedy movie. It lacks the lovey dovey story lines(I wouldn't really call richies e nfatuation with Gina Carbonara love would you? Or him and eddie going up there naked... not love) that make the rest of comedys go from good to crap, it lacks the usual dilemma s that one must overcome in most other comedy movies... unless you count the fact that t hey poisoned the guests and must escape from the guests green vomit as a dilemma thats s imilar to other comedy movies..... No, this movie just sets out and succeeds in doing on e thing AND ONE thing only: Making one laugh. What does one require from comedy movies? Laughter. This movie just piles on laugh after laugh without stuffing up the laughs with serious crap like other comedy movies! Thus I call it the, so far, only perfect comedy m ovie ever made and I will never ever stop watching this beautiful movie! I appluad rick and ade on such fantastic genius!

4. 15 PARK AVENUE is the address "Mithi/Mithali" (Konkona) is in search for from the mov ies beginning. "Prof.Anu" (Shabhana Azmi)is Mithi's extremely caring and loving half sis ter from Mithi's mom's earlier marriage. The movie revolves around these characters and looks into the life of a schizophrenic patient (Mithi). The director tries to explain to the viewer the imaginary world of Mithi, through her continuous blabbering to Anu and ot hers.

Konkona deserves not one but thousands of awards (which I am sure, she will be getting)for this rendition of Mithi in this movie. You can see the look of a pat ient written on her face, by the drooping lips and sleepy eyes, from the first scene its elf. Rahul Bose has done a good job, but has been reduced to one half of the movie in sp ite of his importance in their life.

Watch out for the intense relationships shown between the characters of the movie, Mithi & Anu, Anu & Anu's Mom and between Anu & Sanjiv (Kanwaljit Singh). Shabhana Azmi, as usual has done a riveting performance to b e remembered as the sister, who sacrificed her life for Mithi.

The movie migh t not be your usual Hindi potboiler, but can certainly make people look at the schizophr enic patients in a different light altogether.

As usual, Aparna Sen brings th e movie to a different ending rather than any clichéd ones, we might think off. Hats off to her, for this great movie!!!

5. i was disappointed in this documentary.i thought it would be about the second chess m atch between Grandmaster Garry Kasporov and Deep Blue the supercomputer designed by IBM computer experts to beat any human chess player.Kasparov was and still is,considered the greatest chess player ever.the movie takes us back to 1997 where Kasporov had agreed to

```
have a rematch with "deep Blue" after defeating it 1 year earlier.but instead of focusin
g on the game,it focuses more on what happens before and after.there are snippets of the
game,but not very many.much of the film centers around Kasporov's paranoid obsession tha
t the match was rigged as part of some conspiracy theory and that he lost the match unfa
irly.the movie also includes interviews with people who are not interesting in any way.t
hey even chat with the manager of the building where the match took place.who cares?i al
so found it very dry and slow.ultimately this movie was unsatisfying.this is just my opi
nion,of course.if you like conspiracy theories,this movie might interest you.for people
not into chess or conspiracy theories,this movie would probably have no value.i am a che
ss fan,and i only stuck it out because of that.i give"Game Over:Kasparov and the Machin
e" 4/10
```

Analysis

We will interest in the 1st and 5th documents.

In the first one, while the author enjoyed the movie, it also understands why other people would dislike it, and explains so. The model got stuck on sentence like "most of the people who hate to watch slow movies", without understanding that the dark mood of the movie actually resonates with the reviewer. This is therefore a false negative

The fifth one is a contrario a false positive. The author is actually let down by the movie, which according to him does not do justice to the match between the *greatest* chess player Kasparov. The model misunderstanding all the superlatives to be about the film, while in reality the reviwever explains why he is let down.

In both case, we have respectively bad and good words used in a good and bad context, which tricks the models as it does not have a notion of context.

FastText

Dataset conversion

In [20]:
```python
import re

def preprocessing(doc : str) -> str:
    """
    Lowers and remove punctuations, as asked in subject.
    :param doc: Doc to preprocess.
    :return: doc in lowercase and without punctuation.
    """
    # Found the regex below to remove all punctuation in a string. Neat!
    return re.sub(r'[^\w\s]', '', doc.lower())

def DS_to_FastText(ds : Dataset) -> str:
    """
    Converts the HuggingFace DataSet to a file comprehensible by FastText.
    :param ds: The DataSet object to convert.
    :return: A single str, where each line is a doc prefixed by the labelled class.
    """
    labels = ds.features['label'].names
    converted_lines = [f"__label__{labels[doc['label']]} {preprocessing(doc['text'])}" f
    return '\n'.join(converted_lines)

def write_DS_as_FastText(ds : Dataset, path : str) -> None:
    """
    Uses DS_to_FastText() to convert DataSet then write to a file.
    :param ds: The DataSet object to convert.
    :param path: Where the file should be saved.
```

```
    """
    f = open(path, "wb")
    f.write(DS_to_FastText(ds.shuffle()).encode('utf-8'))
    f.close()
```

In [21]:
```
# Converting and writing dataset
PATH_FASTTEXT_FULL_TRAIN = "datasets/full_train.txt"
PATH_FASTTEXT_FULL_TEST = "datasets/full_test.txt"

write_DS_as_FastText(ds_train, PATH_FASTTEXT_FULL_TRAIN)
write_DS_as_FastText(ds_test, PATH_FASTTEXT_FULL_TEST)
```

```
Loading cached shuffled indices for dataset at C:\Users\Griffures\.cache\huggingface\dat
asets\imdb\plain_text\1.0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f056
1f9a\cache-faf8c2d959925f7e.arrow
Loading cached shuffled indices for dataset at C:\Users\Griffures\.cache\huggingface\dat
asets\imdb\plain_text\1.0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f056
1f9a\cache-e3a051aef336c446.arrow
```

Training

With default hypers

In [22]:
```
from fasttext import FastText

defaultFT = FastText.train_supervised(input = PATH_FASTTEXT_FULL_TRAIN)
```

In [23]:
```
defaultFT_acc = defaultFT.test(PATH_FASTTEXT_FULL_TEST)[1]
print("Accuracy :", defaultFT_acc)
```

```
Accuracy : 0.87552
```

Hypers search functionality

We will first need to split the dataset. Since I already have a function converting the dataset to a text file compatible with FastText, I will not use sklearn's function, but simply use HuggingGace's Slicing API.

In [24]:
```
# Config
PATH_FASTTEXT_SPLIT_TRAIN = "datasets/split_train.txt"
PATH_FASTTEXT_SPLIT_VALID = "datasets/split_valid.txt"
VALID_SPLIT_PCT = 50

# Actual splitting
threshold = VALID_SPLIT_PCT // 2
ds_split_valid = load_dataset('imdb', split=f'train[:{threshold}%]+train[-{threshold}%:]
ds_split_train = load_dataset('imdb', split=f'train[{threshold}%:-{threshold}%]') # Trai

# Dataset are already shuffled inside the conversion function.
write_DS_as_FastText(ds_split_train, PATH_FASTTEXT_SPLIT_TRAIN)
write_DS_as_FastText(ds_split_valid, PATH_FASTTEXT_SPLIT_VALID)
```

```
Reusing dataset imdb (C:\Users\Griffures\.cache\huggingface\datasets\imdb\plain_text\1.
0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f0561f9a)
Reusing dataset imdb (C:\Users\Griffures\.cache\huggingface\datasets\imdb\plain_text\1.
0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f0561f9a)
Loading cached shuffled indices for dataset at C:\Users\Griffures\.cache\huggingface\dat
asets\imdb\plain_text\1.0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f056
1f9a\cache-2543f99785e8eedb.arrow
Loading cached shuffled indices for dataset at C:\Users\Griffures\.cache\huggingface\dat
asets\imdb\plain_text\1.0.0\e3c66f1788a67a89c7058d97ff62b6c30531e05b549de56d3ab91891f056
1f9a\cache-672e9aaa5d21540f.arrow
```

In [25]:
```
# Search
hyperFT = FastText.train_supervised(input=PATH_FASTTEXT_SPLIT_TRAIN, autotuneValidationF
```

In [26]:
```
# Evaluate
hyperFT_acc = hyperFT.test(PATH_FASTTEXT_FULL_TEST)[1]
print("Accuracy :", hyperFT_acc)
```

```
Accuracy : 0.88116
```

Model differences

In [27]:
```
# Attributes

def extract_attributes(model : FastText) -> str:
    return f"""attributes :
  * bucket : {model.bucket}
  * dim : {model.dim}
  * lr : {model.lr}
  * lrUpdateRate : {model.lrUpdateRate}
  * maxn : {model.maxn}
  * minn : {model.minn}
  * wordNgrams : {model.wordNgrams}
  * ws : {model.ws}
  * loss : {model.loss}
  * minCount : {model.minCount}
  * minCountLabel : {model.minCountLabel}
"""
```

In [28]:
```
print("Default model", extract_attributes(defaultFT))
print("Hypertuned model", extract_attributes(hyperFT))
```

```
Default model attributes :
  * bucket : 0
  * dim : 100
  * lr : 0.1
  * lrUpdateRate : 100
  * maxn : 0
  * minn : 0
  * wordNgrams : 1
  * ws : 5
  * loss : loss_name.softmax
  * minCount : 1
  * minCountLabel : 0

Hypertuned model attributes :
  * bucket : 10000000
  * dim : 5
  * lr : 2.257449767122032
  * lrUpdateRate : 100
  * maxn : 0
  * minn : 0
  * wordNgrams : 3
  * ws : 5
  * loss : loss_name.softmax
  * minCount : 1
  * minCountLabel : 0
```

Analysis :

- The hypertuned model is way more aggressive in its learning rate (quintuple the default one)
- Its dimension however, is reduced by an order of magnitude (100 vs 9)

- The hypertuned model work using trigrams, which was hypothesized when working with the NaiveBayes model above.
- minn and maxn are both nullified in both models.

Digging up errors

In order to compare the models, we will use the 5 same sentences as in the Naive Bayesian model.

In [29]:
```
tricky_labels_pred = hyperFT.predict([preprocessing(err) for err in five_errors])[0]
```

In [30]:
```
for i, (error, label) in enumerate(list(zip(five_errors, tricky_labels_pred))):
    print(f"{i + 1}. {label[0][9:].upper()} : {preprocessing(error)}")
```

```
1. NEG : the description of the mood of the movie may be considered as a spoiler  becaus
e there is not much action in factbr br great onebr br is it for my peculiar interest fo
r the dystopias and utopias is it for the atmosphere of the movie or is there some more
magic if yes it is for sure the utmost human onebr br this film is no doubt extremely ar
tisticartificial depends on taste i can imagine most of the people who hate to watch slo
w movies and those of tsai ming liang who i didnt enjoy other times are one of the slowe
st that i know suffering during the movie yes people are unable to slow down and to let
time pass  and to watch it without feeling they waste it one can take this piece as tort
ure or as a therapybr br the topic at the surface the lack of communication  even if we
live in rabbit cages  one next to each other  but not really together people are tired s
ick of something and unable to describe it  just dont want to meet touch talk confront t
he others like if they had disappeared the big block of flats looks void and the rain fa
lling constantly evokes the strange melancholy inside and sometimes it must be something
abnormal unexpected some unwanted decay as a hole in the floor of concrete  that allows
us to reach each otherbr br one of the possible ways to look at it is this dont survey t
he inner world of the characters  consider the whole moviespace to be inside of yourself
and ask  why is it there where could these depressive states and moods come form is ther
e a place for them they dont have a right to be here and search for the answers if you n
eed them among the walls and halls of the block  instead of inside hardly transparent mi
nd of a manbr br the key to understand is nottounderstand  to let a movie borrow us  as
a subject of study  inside itself  and at the end safely return us to our more colorful
and normal looking realitybr br then maybe you will reach  like me  the feeling of real
possible nonpathetic hope that in core we are still humans and this state of mind can he
lp one much to live in this world
2. POS : the trailer to this film focused so much on the chain of course because its so
sensational that it missed most of the movie which is about a developing although rather
simply drawn relationship between lazarus and rae as they attempt to recover from their
past pains with each otherbr br of course with the premise of a nymphomaniac in chains i
ts no surprise that theres plenty of implied sex involved however at its core black snak
e moan is a basic tale of redemption and the healing power of helping another person alo
ng maybe its just me though but i think poor lazarus shouldve had his story focused on m
ore hes a hurting man after his wife leaves him but we never fully see how helping rae r
esolve her past pains heals him too its just implied that it doesin essence he plays the
wizard that helps the young rae overcome her curse through a big ol chain and some blues
br br i like the story but i wish it were a bit more even and didnt have to rely on the
sensational the side characters were fairly decent if simple and i liked the music the a
cting was good enough although i cant be certain if the rae character is fully believabl
e but that might just be my naivetybr br all in all i liked the film but i wasnt compell
ed by it maybe its that im too critical but the story seems a little too convenient to b
e fully believable and so while it all seemed very cool i could never truly buy it the c
hain thing was a little too farfetched for me still this can provide some entertainment
for those looking for dramatic redemption stories with a shot of the blues 710
3. POS : i havent written a review on here in ages but rewatching all of bottom tv show
live shows and this i felt i had to make my views on this movie known it is i feel the p
erfect comedy movie it lacks the lovey dovey story linesi wouldnt really call richies en
fatuation with gina carbonara love would you or him and eddie going up there naked not l
ove that make the rest of comedys go from good to crap it lacks the usual dilemmas that
one must overcome in most other comedy movies unless you count the fact that they poison
ed the guests and must escape from the guests green vomit as a dilemma thats similar to
```

```
other comedy movies no this movie just sets out and succeeds in doing one thing and one
thing only making one laugh what does one require from comedy movies laughter this movie
just piles on laugh after laugh without stuffing up the laughs with serious crap like ot
her comedy movies thus i call it the so far only perfect comedy movie ever made and i wi
ll never ever stop watching this beautiful movie i appluad rick and ade on such fantasti
c genius
4. POS : 15 park avenue is the address mithimithali konkona is in search for from the mo
vies beginning profanu shabhana azmiis mithis extremely caring and loving half sister fr
om mithis moms earlier marriage the movie revolves around these characters and looks int
o the life of a schizophrenic patient mithi the director tries to explain to the viewer
the imaginary world of mithi through her continuous blabbering to anu and others br br k
onkona deserves not one but thousands of awards which i am sure she will be gettingfor t
his rendition of mithi in this movie you can see the look of a patient written on her fa
ce by the drooping lips and sleepy eyes from the first scene itself rahul bose has done
a good job but has been reduced to one half of the movie in spite of his importance in t
heir lifebr br watch out for the intense relationships shown between the characters of t
he movie mithi  anu anu  anus mom and between anu  sanjiv kanwaljit singh shabhana azmi
as usual has done a riveting performance to be remembered as the sister who sacrificed h
er life for mithibr br the movie might not be your usual hindi potboiler but can certain
ly make people look at the schizophrenic patients in a different light altogetherbr br a
s usual aparna sen brings the movie to a different ending rather than any clichéd ones w
e might think off hats off to her for this great movie
5. NEG : i was disappointed in this documentaryi thought it would be about the second ch
ess match between grandmaster garry kasporov and deep blue the supercomputer designed by
ibm computer experts to beat any human chess playerkasparov was and still isconsidered t
he greatest chess player everthe movie takes us back to 1997 where kasporov had agreed t
o have a rematch with deep blue after defeating it 1 year earlierbut instead of focusing
on the gameit focuses more on what happens before and afterthere are snippets of the gam
ebut not very manymuch of the film centers around kasporovs paranoid obsession that the
match was rigged as part of some conspiracy theory and that he lost the match unfairlyth
e movie also includes interviews with people who are not interesting in any waythey even
chat with the manager of the building where the match took placewho caresi also found it
very dry and slowultimately this movie was unsatisfyingthis is just my opinionof coursei
f you like conspiracy theoriesthis movie might interest youfor people not into chess or
conspiracy theoriesthis movie would probably have no valuei am a chess fanand i only stu
ck it out because of thati givegame overkasparov and the machine 410
```

Analysis :

We see that most of the tricky cases failed by the Naive Bayesian model were correctly guessed by FastText. However, it still got lead astray int he first one by the explanation of how the reviewer may understand why *others* may have disliked the movie (but how he personally enjoyed it nonetheless).

To be fair to the model this example can be considered tricky, as the reviewer employs many subjunctives, so tracking context is difficult. Even more so that author says how the movie touched him, but did not say outright that he *liked* it (as human, we have the inner knowledge that art has the goal of resonating, one way or another, with the observer. However again, FastText lacks such context).

MINN and MAXN nullified ?

Minn and Maxn are filter applied on N-Grams to filter out words too small (minn) or too big (maxn).

Them both being nullified signify that absolutely no words are being filtered out, the models takes into account every feature.

This can be understood, as in english, words have very varied length (especially with 3-grams in this model's case). While prepositions tends to be shorter and carry less information, so does other important words.

For example, "not" is only three letters and changes completely the sense of the following proposition, "like" is one of the main features of our classifier and is only 4 letters long. On the other hand, with have 3 grams

like "actors played perfectly", 24 letters long.

The model therefore takes all those words into account and do not discriminate by length.

Theoretical questions

Answer the following questions.

---

1. Explain with your own words, using a short paragraph for each, what are:
    - Phonetics and phonology
    - Morphology and syntax
    - Semantics and pragmatics

Phonetics and phonology : Phonetics are linked to human sounds, as of which could be found inside a phonetic alphabet (mostly used therefore in oral language processing). Those are divided between production, transmission, and perception category. Phonology, on the other end, comes after phonetics. It is link to those sounds inside a language context (or multiple ones), in order to form syllables and words (or language's base elements).

Morphology and Syntax : Morphology comes after phonology, and looks at words and their composition (stems, prefixes, suffices, etc) and more generally how words relate to each other. Syntax, still one step further, look into the relation between *words* to establish sentences (with context, meaning). Grammatical relations starts from here.

Semantics and pragmatics : We top off our layers with the former and the latter, in this order. Semantics is the study of meaning through the relation if words and sentences, in a larger sense. Pragmatics view this same meaning through the lens of social context (no you could *'eat an elephant*', you are just very hungry).

---

1. What is the difference between stemming and lemmatization?
    - How do they both work?
    - What are the pros and cons of both methods?

Stemming is used to get back the *stem* of the words (its root). That way, 'movie' becomes 'movy' as we saw *(ie -> y)*, or 'skipping' become 'skip' (advanced stemming will sometimes keep suffixes, instead transforming it to 'skip' and '#ing', in order to keep the information about continuity). It is purely algorithmical.

Lemmatization, as its name implies, include a collection of *lemme*. Those are hand crafted rules that helps the model link words together. For example, we indicate to the model that *"were" == "was" == "'s"* and should be handled as *"be"*.

Lemmatization tends to have better results than Stemming, however the latter is way easier to implement, and also cheaper to run resource wise and time wise. On older models, such preprocessing tends to worsen accuracy instead.

---

1. On logistic regression:
    - How does stochastic gradient descent work?
    - What is the role of the learning rate?

- Will it always find the global minimum?

Stochastic Gradient Descent works by deriving our modelled function and modifying our inputs in the direction of our obtained gradient, in order to gradually converge towards the modelled function minimum.

Learning rate correspond to how big of a step we take at each iteration. Proper learning rate is very important, as one too low would make the model take ages to converge, or even get stuck inside a local minimum. We could also totally step over the minima in case of a learning to high.

A mechanism of *heat* is sometimes used to circumvent those issues : The learning rate start high in order to quickly explore the domain of the modelled function, then the learning rate slows down (as would heat cool off) in order to get more precise results.

The global (absolute) minimum is absolutely not guaranteed, as getting stuck inside a local minimum is very common (plus the example above).

---

1. What problems does TF-iDF try to solve?
    - What the is the TF part for?
    - What is the iDF part for?

TF-iDF is used to weight features (N-grams) inside a document corpus (it is applied for each word for each document).

The **TF** (term frequency) part weights favorably a term that is used multiple time in the current document (*if it is repeated, then it is an important term in this specific document*).

It is most simply a simple fraction (occurrences of a specific feature in the doc divided by the total number of features inside said doc).

The **iDF** (inverse document frequency) part weights unfavorably a word that is used in many document of the corpus (*if this term is important for every document, then it does not give me information this specific one*).

It is the log of the number of documents (N) divided by the document frequency of the studied term (DF). As DF approaches N (the term appears in every document), the fraction becomes 1, and the log lean towards 0, devaluing the weight of the term.

TF-iDF is simply the product of those two parts.

---

1. Summarize how the skip-gram method of Word2Vec works using a couple of paragraphs.
    - How does it uses the fact that *two words appearing in similar contexts are likely to have similar meanings*?

The **skip-gram method of Word2Vec** works in reverse of models like *CBOW*. Instead of providing *context* (surrounding words) and then having to guess the center words, it provides the center words and then tries to guess the surrounding ones. Essentially creating proposition.

To do so, it does not use *one-hot encoding* but *continuous vectors* (a.k.a. embeddings). That way, words that are closer in meaning are also closer in an euclidean space.

If two different words can be found with the same surrounding words, then the model deduce that they are closer in meaning, bringer also literally closer in the embedding space.