

Table of Contents

1. Datasets building
2. Translation
3. Normization
4. Sentiment analysis (step 1)
5. Sentiment analysis (step 2 & 3)

1. Datasets building

Dependencies

Entrée [5]:

```

1 import pandas as pd
2 import os
3 import glob
4 from datasets import load_dataset
5 from pandas import DataFrame
6 import pickle

```

English Movies

Entrée [94]:

```

1 # glob allows to merge all csv files in the folder
2 path = os.getcwd()
3 csv_files = glob.glob(os.path.join(path, "*.csv"))
4
5 for f in csv_files:
6
7     df = pd.read_csv(f, usecols=["rating", "review"], delimiter=',')
8
9     # Delete reviews longer than 2000 characters
10    df=df[df["review"].str.len() < 2000]
11
12    # Categorize negative reviews
13    df4 = df[df["rating"] == "4"]
14    df3 = df[df["rating"] == "3"]
15    df2 = df[df["rating"] == "2"]
16    df1 = df[df["rating"] == "1"]
17    frames1 = [df1, df2, df3, df4]
18    db1=pd.concat(frames1)
19    db1["rating"]="negative"
20
21    ## Categorize positive reviews
22    df10 = df[df["rating"] == "10"]
23    df9 = df[df["rating"] == "9"]
24    df8 = df[df["rating"] == "8"]
25    frames2 = [df8, df9, df10]
26    db2=pd.concat(frames2)
27    db2["rating"]="positive"
28
29    #Balance positive and negative reviews
30    if len(db1) != len(db2):
31        N = max(len(db1),len(db2))-min(len(db1),len(db2))
32
33        if len(db1) > len(db2):
34            dt = db1.drop(db1[db1['rating'].eq("negative")].sample(N).index)
35            data=pd.concat([db2,dt])
36
37        if len(db1) < len(db2):
38            dt = db2.drop(db2[db2['rating'].eq("positive")].sample(N).index)
39            data=pd.concat([db1,dt])
40
41    else:
42        data=pd.concat([db1,db2])
43
44    data = data.replace(to_replace=[r"<br /><br />"], value=[" "], regex=True)

```

Entrée [23]:

```
1 data = data.dropna()
2 data = data.drop_duplicates()
3 data = data.reset_index(drop=True)
4 data
```

Out[23]:

	rating	review
0	negative	A great example of how didacticism bores rather than inspires, regurgitates rather than re-imagines. There's a clear political message, and the unabashed way which it is presented won't change any minds, but at the least may spark political debate. Though its singular treatment preaching the current "hip" left-wing social platforms without nuance does nothing to elucidate any new ideas or attempt to shed new truths on issues that allow people to re-evaluate their current ideological stances. As such, its effect is similar to that of the person talking politics at the cocktail party who relies on volume rather than substance in order to bolster their (or more accurately their party's) talking points. All in all, though there are some witty scenes and stereotypical romance & coming of age story lines, the overpowering pedantic verbal diarrhea exposes it as a thinly veiled political commercial rather than a great piece of art. The ratings, therefore, represent the extent to which the au...
1	negative	There was not even one funny content, sloppy dialogs, and the story totally flat. I doubt it was a comedy. I wouldn't suggest it for children, as the plot is hardly understandable even adults. By understandable I mean exciting. This movie just happened. No memories left. No magic left. It's hard to write anything cause the movie was empty. Disappointing. I love Lion the King, Tangled, WALL-E, Hercules, but this movie is a disgrace! It is full of humor for adults, crappy humor by the way. Children sitting behind me asked their father: "Can we go home, please", and it was just 30 minutes of movie. They started asking repeatedly after one hour. I totally agreed with them. I wanted to go home and watch Lion the King. I wanted my evening back. Nice marketing Disney! Sadly. You won't catch me again. Is it the end of animation for children and people with young hearts?
2	negative	Zootopia is Disney's latest propaganda piece for the fascist police state. They don't even bother to be subtle any more; nothing is left to the imagination. It's all here - the 99 percent vs the one percent, cops who just want to serve and protect you (and BTW they're definitely NOT serving the one percent), and ubiquitous public cameras that help solve crimes. And the punch line? The one percent are really your friends - it's those ingrate 99-percenters who are the real criminals. This film is wonderful way to encourage impressionable young children to conform into little statist robots who will gladly empty their minds and hearts to serve as a cog in the machine of the modern lie. And now, a message from your master...

Entrée []:

```
1 data = data.sort_values(by=["rating"], ascending=False)
2 data = data.drop(tweet[tweet['rating'].eq("positive")].sample(410).index)
3 data = data.drop(tweet[tweet['rating'].eq("negative")].sample(410).index)
```

Entrée [70]:

```
1 data["rating"].value_counts()
```

Out[70]:

negative 80000
positive 80000
Name: rating, dtype: int64

Entrée [71]:

```
1 pickle.dump(data, open("english_to_french_film.dataset", "wb"))
```

French Movies

Entrée []:

```
1 dataset = load_dataset('allocine')
2
3 columns_to_return = ['label', 'review']
4 dataset.set_format(type='pandas', columns=columns_to_return)
5 data=pd.DataFrame.from_dict(dataset)
6
7 pickle.dump(data, open("french_to_english_film.dataset", "wb"))
```

Amazon Products

Entrée []:

```
1 # Products in French
2 dataset = load_dataset('amazon_reviews_multi', 'fr')
3 db=dataset['review_body']
4 db1=dataset['stars']
5
6 df = DataFrame (db)
7 df1 = DataFrame (db1)
8
9 data=pd.concat([df1,df],axis='columns')
10 data.columns=["rating", "review"]
11
12 pickle.dump(data, open("english_to_french_product.dataset", "wb"))
13
14
15 # Products in English
16 dataset = load_dataset('amazon_reviews_multi', 'en')
17 db=dataset['review_body']
18 db1=dataset['stars']
19
20 df = DataFrame (db)
21 df1 = DataFrame (db1)
22
23 data=pd.concat([df1,df],axis='columns')
24 data.columns=["rating", "review"]
25
26 pickle.dump(data, open("french_to_english_product.dataset", "wb"))
```

Merge Review and Translation

```
Entrée [6]: 1 df=pickle.load(open("french_to_english_product.dataset", "rb"))
2 df
```

Out[6]:

	rating	review	translation	review_proc	review_proc+lemma	translation_proc	translation_proc+lemma
0	1	A déconseiller - Article n'a fonctionné qu'une...	A discouragement - article Na worked that once...	déconseiller Article fonctionné une fois recom...	déconseiller Article fonctionner un fois recom...	discouragement article worked that once not re...	discouragement article work that once not reco...
1	1	Si vous voulez être déçu achetez le produit ! ...	If you want to be disappointed buy the product...	Si vous voulez être déçu achetez produit bout ...	si vous vouloir être décevoir achetez produire...	If you want disappointed buy the product After...	if -PRON- want disappoint buy the product afte...
2	1	Écran de mauvaise qualité, car il s'use en peu...	Screen of poor quality because it suses in a s...	cran mauvaise qualité car use peu temps croche...	cran mauvais qualité car use peu temps croche ...	Screen poor quality because suses short time a...	screen poor quality because suse short time an...
3	1	Cet engin ne sert à rien les sons sont pourris...	This machine does not serve the sounds are rot...	Cet engin sert rien les sons sont pourris les ...	ce engin servir rien le son être pourrir le so...	This machine does not serve the sounds are rot...	this machine do not serve the sound be rotten ...
4	1	Très beau produit mais la grue n'a pas fonctio...	Very nice product but the crane did not work v...	Très beau produit mais grue pas fonctionné très...	très beau produire mais grue pas fonctionner t...	Very nice product but the crane did not work v...	very nice product but the crane do not work ve...
...	...	...	...	...	...	...	...

```
Entrée [ ]: 1 df1=pickle.load(open("french_to_english_product.dict", "rb"))
```

```
Entrée [ ]: 1 # Create a dataframe for the translation
2 df2 = pd.DataFrame.from_dict(df1, orient='index', columns={"translation"})
3
4 # Merge dataframes
5 data = pd.concat([df,df2],axis='columns')
6
7 # Renaming Labels
8 data.loc[(data.rating == "negative"), "rating"] = "0"
9 data.loc[(data.rating == "positive"), "rating"] = "1"
10
11 # Reordering if needed
12 data = data.sort_values(by=["rating"])
13 data = data.reset_index(drop=True)
14
15 #Save finished dataset
16 pickle.dump(data, open("english_to_french_film.dataset", "wb"))
```

2. Translation

Dependencies and data

```
Entrée [1]: 1 import pickle
2 import pandas as pd
3 from pygoogletranslation import Translator
4
5 df=pickle.load(open("english_film.dataset", "rb"))
6 save_name = "english_to_french_film"
7 df1=df["review"]
```

Traduction

```
Entrée [2]: 1 # Check if a translation file already exists and imports it, otherwise it creates it.
2 #Allows to translate bit by bit
3
4 first = False
5 if first :
6     dict_trad={}
7
8     translator = Translator()
9     result = translator.translate(df1[0], src='en', dest='fr')
10    dict_trad[0]=result.text
11
12 else :
13     dict_trad=pickle.load(open("english_to_french_film.dict", "rb"))
14
15 len(dict_trad)
```

Out[2]: 154674

```
Entrée [7]: 1 # Check last translation
2
3 df1.iloc[len(dict_trad)-1]
```

Out[7]: "NOES sets the mark for horror of the 'teen slasher' genre in my book. Forget *Scream*, this film came far before and has a far more interesting story. Englund is excellent as nightmarish bad guy Freddie Krueger, and while there have been many sequels, and Krueger has now become a cult icon, it is only within this (ok and maybe Wes Craven's *New Nightmare*) that it really shows just how good a bad guy he is. Terrifying, but nevertheless great."

```
Entrée [8]: 1 dict_trad[len(dict_trad)-1]
```

Out[8]: "Noes définit la marque pour l'horreur du genre Teen Slasher dans mon livre. Oubliez *Scream*, ce film est arrivé loin avant et a une histoire beaucoup plus intéressante. ENGLUND est excellent comme le méchant méchant Guy Freddie Krueger, et bien qu'il y ait eu de nombreuses suites, et Krueger est devenue une icône de culte, elle ne se trouve que dans ceci (OK et peut-être que Wes Craven's *New Nightmare*), qu'il montre vraiment à quel point mec il est. Terrifiant, mais néanmoins génial."

```
Entrée [2]: 1 # Translate from last element using Google Traduction API
2
3 def Traduction(original,trad):
4
5     translator = Translator()
6
7     for i in range(len(trad),len(original),1):
8         result = translator.translate(original[i], src='en', dest='fr')
9         trad[i]=result.text
10        print('Running for review nb : '+str(i)+'/'+str(len(original)),end='\r')
```

```
Entrée [1]: 1 Traduction(df1,dict_trad)
```

Running for review nb : 159999/160000

```
Entrée [11]: 1 # Save translation using pickle
2
3 print(len(dict_trad))
4 save = True
5 if save :
6     pickle.dump(dict_trad, open(save_name+".dict", "wb"))
```

160000

3. Normalization

Dependencies

Entrée [2]:

```
1 import pandas as pd
2 import numpy as np
3 #pd.set_option('display.max_rows', 50)
4 #pd.set_option('display.max_colwidth', 1000)
5 from nltk.tokenize import RegexpTokenizer
6 import pickle
7 import spacy
8 import re
9 import string
10 import seaborn as sns
11 import matplotlib.pyplot as plt
12 import time
13 from wordcloud import WordCloud
```

Data

Entrée [5]:

```
1 data=pickle.load(open("english_to_french_film.dataset", "rb"))
2 data_name = "En_to_fr_movie"
3 data
```

Out[5]:

	rating	review	translation	review_proc	review_proc+lemma	translation_proc	translation_proc+lemma
0	0	A great example of how didacticism bores rather ...	Un excellent exemple de la façon dont la didac...	great example how didacticism bores rather than ...	great example how didacticism bore rather than i...	excellent exemple façon dont didactisme est pl...	excellent exemple façon dont didactisme être p...
1	0	There was not even one funny content, sloppy d...	Il n'y avait même pas un contenu drôle, des di...	There was not even one funny content sloppy di...	there be not even one funny content sloppy dia...	avait même pas contenu des dialogues clés hist...	avoir même pas contenu de dialogue clé histo...
2	0	Zootopia is Disney's latest propaganda piece f...	La Zootopie est Disneys Dernière pièce de prop...	Zootopia Disney latest propaganda piece for th...	Zootopia Disney late propaganda piece for the ...	Zootopie est Disneys Dernière pièce propagande...	Zootopie être disneys dernier pièce propagande...
3	0	Disney does not make movies for children anymo...	Disney ne fait plus de films pour enfants. Ils...	Disney does not make movies for children anymo...	Disney do not make movie for child anymore -PR...	Disney fait plus films pour enfants Ils font d...	disney faire plus film pour enfant il faire un...
4	0	Zootopia. A popular and buzz worthy film. But ...	Zootopie. Un film de buzz et buzz. Mais pourqu...	Zootopia popular and buzz worthy film But why ...	Zootopia popular and buzz worthy film but why ...	Zootopie film buzz buzz Mais pourquoi Cela rie...	Zootopie film buzz buzz mais pourquoi cela rie...
...	...	...	...	...	...	...	...

Normalization

```

Entrée [4]: 1 # Create the second and third version of normalization consisting in the suppression of web page, mention and hashtags,
2 # everything except letters in the alphabet of the given language et small words except some exceptions
3
4 # The suppression of the stopwords is (if needed) done just before the vectorisation
5
6 English = True
7
8 if English:
9     data['review_proc'] = data['review'].replace(to_replace=[r"http\S+|\S+|\#\S+"], value="", regex=True)
10    data['review_proc'] = data['review_proc'].str.replace("[^a-zA-Z]", " ")
11    data['review_proc'] = data['review_proc'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2 or
12                                w=="no" or w=="so" or w=="So" or w=="No" or w=="If" or w=="if"]))
13
14    load_model = spacy.load('en_core_web_sm')
15
16    tokenized0 = [None]*len(data)
17    for i in range(len(data['review_proc'])):
18        doc = load_model(data['review_proc'].iloc[i])
19        tokenized0[i] = " ".join([token.lemma_ for token in doc])
20        if i%1600==0:
21            print("Obs %.0f -----" % (i), end="\r")
22
23    data['review_proc+lemma'] = tokenized0
24
25    data['translation_proc'] = data['translation'].replace(to_replace=[r"http\S+|\S+|\#\S+"], value="", regex=True)
26    data['translation_proc'] = data['translation_proc'].str.replace("[^a-zA-Z-éèàèçù]", " ")
27    data['translation_proc'] = data['translation_proc'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2 or
28                                w=="si" or w=="ni" or w=="Si" or w=="Ni"]))
29
30    load_model = spacy.load('fr_core_news_sm')
31
32    tokenized1 = [None]*len(data)
33    for i in range(len(data['translation_proc'])):
34        doc = load_model(data['translation_proc'].iloc[i])
35        tokenized1[i] = " ".join([token.lemma_ for token in doc])
36        if i%1600==0:
37            print("Obs %.0f -----" % (i), end="\r")
38
39    data['translation_proc+lemma'] = tokenized1
40
41 else:
42    data['review_proc'] = data['review'].replace(to_replace=[r"http\S+|\S+|\#\S+"], value="", regex=True)
43    data['review_proc'] = data['review_proc'].str.replace("[^a-zA-Z-éèàèçù]", " ")
44    data['translation_proc'] = data['translation_proc'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2 or
45                                w=="si" or w=="ni" or w=="Si" or w=="Ni"]))
46
47    load_model = spacy.load('fr_core_news_sm')
48
49    tokenized0 = [None]*len(data)
50    for i in range(len(data['review_proc'])):
51        doc = load_model(data['review_proc'].iloc[i])
52        tokenized0[i] = " ".join([token.lemma_ for token in doc]) #if not token.lemma_ in all_stopwords
53        if i%1600==0:
54            print("Obs %.0f -----" % (i), end="\r")
55    data['review_proc+lemma'] = tokenized0
56
57    data['translation_proc'] = data['translation'].replace(to_replace=[r"http\S+|\S+|\#\S+"], value="", regex=True)
58    data['translation_proc'] = data['translation_proc'].str.replace("[^a-zA-Z]", " ")
59    data['review_proc'] = data['review_proc'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2 or
60                                w=="no" or w=="so" or w=="So" or w=="No" or w=="If" or w=="if"]))
61
62    load_model = spacy.load('en_core_web_sm')
63
64    tokenized1 = [None]*len(data)
65    for i in range(len(data['translation_proc'])):
66        doc = load_model(data['translation_proc'].iloc[i])
67        tokenized1[i] = " ".join([token.lemma_ for token in doc]) #if not token.lemma_ in all_stopwords
68        if i%1600==0:
69            print("Obs %.0f -----" % (i), end="\r")
70
71    data['translation_proc+lemma'] = tokenized1

```

Obs 158400 -----

Entrée [74]:

1

data

Out[74]:

	rating	review	translation	review_proc	review_proc+lemma	translation_proc	translation_proc+lemma
0	0	A great example of how didacticism bores rather ...	Un excellent exemple de la façon dont la didac...	great example how didacticism bores rather than ...	great example how didacticism bore rather than i...	excellent exemple façon dont didactisme est pl...	excellent exemple façon dont didactisme être p...
1	0	There was not even one funny content, sloppy d...	Il n'y avait même pas un contenu drôle, des di...	There was not even one funny content sloppy di...	there be not even one funny content sloppy dia...	avait même pas contenu des dialogues clés hist...	avoir même pas contenu de dialogue clé histor...
2	0	Zootopia is Disney's latest propaganda piece f...	La Zootopie est Disneys Dernière pièce de prop...	Zootopia Disney latest propaganda piece for th...	Zootopia Disney late propaganda piece for the ...	Zootopie est Disneys Dernière pièce propagande...	Zootopie être disneys dernier pièce propagande...
3	0	Disney does not make movies for children anymo...	Disney ne fait plus de films pour enfants. Ils...	Disney does not make movies for children anymo...	Disney do not make movie for child anymore -PR...	Disney fait plus films pour enfants Ils font d...	disney faire plus film pour enfant il faire un...
4	0	Zootopia. A popular and buzz worthy film. But ...	Zootopie. Un film de buzz et buzz. Mais pourqu...	Zootopia popular and buzz worthy film But why ...	Zootopia popular and buzz worthy film but why ...	Zootopie film buzz buzz Mais pourquoi Cela rie...	Zootopie film buzz buzz mais pourquoi cela rie...
...	...	...	...	...	...	...	...
159995	1	If you've ever seen a UFO, this will chill you...	Si vous avez déjà vu un OVNI, cela refroidira ...	If you ever seen UFO this will chill your bloo...	if -PRON- ever see UFO this will chill -PRON- ...	Si vous avez déjà OVNI cela refroidira votre s...	si vous avoir déjà ovni cela refroidir votre s...
159996	1	So when I first heard about this movie, I thou...	Alors, quand j'ai entendu parler de ce film, j...	So when first heard about this movie thought l...	so when first hear about this movie thought lo...	Alors quand entendu parler film pensais que av...	alors quand entendre parler film pensai que av...
159997	1	I absolutely enjoyed this one. As a person who...	J'ai vraiment apprécié celui-ci.En tant que pe...	absolutely enjoyed this one person who has see...	absolutely enjoy this one person who have see ...	vraiment apprécié celui tant que personne qui ...	vraiment apprécier celui tant que personne qui...
159998	1	I haven't reviewed anything for a very long ti...	Je n'ai pas examiné quelque chose depuis très ...	haven reviewed anything for very long time des...	haven review anything for very long time despi...	pas examiné quelque chose depuis très longtemp...	pas examiner quelque chose depuis très longtem...
159999	1	Michelle(Mary Elizabeth Winstead) is in a car ...	Michelle (Mary Elizabeth Winsad) est dans une ...	Michelle Mary Elizabeth Winstead car wreck and...	Michelle Mary Elizabeth Winstead car wreck and...	Michelle Mary Elizabeth Winsad est dans une ép...	Michelle Mary Elizabeth Winsad être dans un ép...

160000 rows × 7 columns

Entrée [10]:

1

Saves the preprocessed dataset

2

pickle.dump(data, open("english_to_french_film.dataset", "wb"))

Wordclouds and exploration

```

1 #Definition
2 column = "review"
3
4 #Preprocessing
5 data[column] = data[column].replace(to_replace=["-PRON-", "être"], value=["", ""], regex=True)
6 data[column] = data[column].apply(lambda x: ' '.join([w for w in x.split() if len(w)>2]))
7 all_words = ' '.join([text for text in data[column]])
8
9 #Plotting
10 wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_words)
11
12 plt.figure(figsize=(15, 12))
13 plt.imshow(wordcloud, interpolation="bilinear")
14 plt.axis('off')
15 plt.savefig("plot/"+data_name+"/wordcloud "+column+".png", bbox_inches="tight")
16 plt.show()

```



4. Performing Sentiment Analysis using Text Classification (Step 1)

Dependencies

```
Entrée [2]: 1 import pandas as pd
2 import numpy as np
3 #pd.set_option('display.max_rows', 50)
4 #pd.set_option('display.max_colwidth', 1000)
5 from sklearn.feature_extraction.text import CountVectorizer
6 from sklearn.feature_extraction.text import TfidfVectorizer
7 from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, cross_validate
8 from sklearn.naive_bayes import MultinomialNB
9 from sklearn.linear_model import LogisticRegression
10 from sklearn.pipeline import Pipeline
11 from sklearn import metrics
12 import pickle
13 import spacy
14 import seaborn as sns
15 import matplotlib.pyplot as plt
16 import time
17 import warnings
18 from sklearn import exceptions
19 warnings.filterwarnings("ignore", category=exceptions.ConvergenceWarning)
```

Functions

```
Entrée [3]: 1 #Takes the choosen values for hyperameter and a classifier as parameters
2 #and perform a gridsearch with a scoring based on accuracy
3 def grid(param_grid,classifier):
4     start = time.time()
5     grid_clf = GridSearchCV(classifier, param_grid = param_grid, scoring = 'accuracy', n_jobs=4,verbose=4, cv=cv)
6     grid_clf.fit(bow, y=Y)
7     end = time.time()
8     print("Best parameters = ", grid_clf.best_params_, " grid time : ", round((end - start)/60, 4))
9     return(grid_clf.best_params_)
```

```
Entrée [5]: 1 #Takes features and the target with a classifier as parameters
2 #and perform the split, training and evaluation (step 1)
3 def train (classifier,X0,Y0):
4     start = time.time()
5     matrix = np.zeros((n,n))
6     accuracy = 0
7     run = 0
8     for train_ix, test_ix in CV.split(X0, Y0):
9         run+=1
10        xtrain, xtest = X0[train_ix, :], X0[test_ix, :]
11        ytrain, ytest = Y0[train_ix], Y0[test_ix]
12
13        #training
14        classifier.fit(xtrain,ytrain)
15
16        # Predicting with a test set
17        predicted = classifier.predict(xtest)
18
19        # Accuracy
20        a = metrics.accuracy_score(ytest, predicted)
21        print("Model Accuracy ",run," : ", a)
22        accuracy = accuracy + a
23
24        #Confusion matrix
25        m = metrics.confusion_matrix(ytest, predicted)
26        matrix = matrix + m
27
28    end = time.time()
29    print("fit time : ", round((end - start)/60, 4))
30    return matrix,accuracy/cv
```

Entrée [6]:

```
1 #Takes the confusion matrix, accuracy and a name as parameters and plot the matrix with all information
2 def plot_matrix (matrix,accuracy,name):
3     fig, ax = plt.subplots()
4     fig.set_size_inches(9, 9)
5     if n == 2:
6         cmd = metrics.ConfusionMatrixDisplay(matrix, display_labels=["Positive","Negative"])
7         ax.text(-0.8,2.1, "Accuracy = "+str(round(accuracy,4))+ "\n"+legend+"\nFeatures = "+str(bow.shape[1])+
8             " Elements = "+str(bow.nnz), fontsize=16)
9     else:
10        cmd = metrics.ConfusionMatrixDisplay(matrix, display_labels=["1","2","3","4","5"])
11        ax.text(-0.5,6, "Accuracy = "+str(round(accuracy,4))+ "\n"+legend+"\nFeatures = "+str(bow.shape[1])+
12            " Elements = "+str(bow.nnz), fontsize=16)
13
14    plt.rc('font', size=18);plt.rc('axes', titlesize=20);plt.rc('axes', labels=14);
15    plt.rc('xtick', labels=12);plt.rc('ytick', labels=12)
16    cmd.plot(ax=ax, cmap="magma_r", values_format='g')
17    ax.set_title(str(name), fontsize=22)
18    #plt.savefig("pLot/"+data_name+" "+ name+"_matrix.png", bbox_inches="tight")
```

Data

Entrée [7]:

```
1 data=pickle.load(open("french_to_english_product.dataset", "rb"))
2 data
```

Out[7]:

	rating	review	translation	review_proc	review_proc+lemma	translation_proc	translation_proc+lemma
0	1	A déconseiller - Article n'a fonctionné qu'une...	A discouragement - article Na worked that once...	déconseiller Article fonctionné une fois recom...	déconseiller Article fonctionner un fois recom...	discouragement article worked that once not re...	discouragement article work that once not reco...
1	1	Si vous voulez être déçu achetez le produit ! ...	If you want to be disappointed buy the product...	Si vous voulez être déçu achetez produit bout ...	si vous vouloir être décevoir achetez produire...	If you want disappointed buy the product After...	if -PRON- want disappoint buy the product afte...
2	1	Écran de mauvaise qualité, car il s'use en peu...	Screen of poor quality because it suses in a s...	cran mauvaise qualité car use peu temps croche...	cran mauvais qualité car use peu temps croche ...	Screen poor quality because suses short time a...	screen poor quality because suse short time an...
3	1	Cet engin ne sert à rien les sons sont pourris...	This machine does not serve the sounds are rot...	Cet engin sert rien les sons sont pourris les ...	ce engin servir rien le son être pourrir le so...	This machine does not serve the sounds are rot...	this machine do not serve the sound be rotten ...
4	1	Très beau produit mais la grue n'a pas fonctio...	Very nice product but the crane did not work v...	Très beau produit mais grue pas fonctionné très...	très beau produire mais grue pas fonctionner t...	Very nice product but the crane did not work v...	very nice product but the crane do not work ve...
...	...	...	...	...	...	...	...

Entrée [8]:

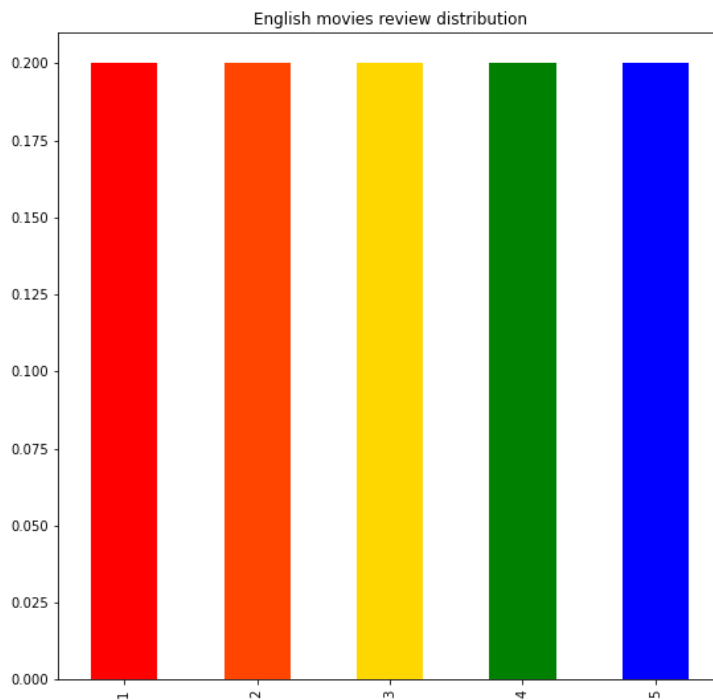
```
1 #Define sample number, fold number for the cross validation and features and target columns
2 cv = 5
3 CV = StratifiedKFold(n_splits=cv, shuffle=True, random_state=1)
4 n = len(data['rating'].value_counts())
5 Y=data['rating']
```

```

Entrée [14]: 1 #Plot the distribution of the target value
2 if n == 2:
3     data["rating"].value_counts(normalize=True).plot(kind='bar', figsize=(9, 9), color=['green', 'orangered'])
4     ax1 = plt.subplot()
5     ax1.set_xticklabels(["Positif", "Negatif"])
6 else:
7     data["rating"].value_counts(normalize=True).plot(kind='bar', figsize=(9, 9),
8                                                         color=['red', 'orangered', 'gold', 'green', 'blue'])
9 plt.title("English movies review distribution")
10 plt.savefig("plot/En_to_fr_movie/distribution.png", bbox_inches="tight")

```

Out[14]: Text(0.5, 1.0, 'English movies review distribution')



Vectorization, training and results

```

Entrée [ ]: 1 #Define dataset and language
2 dataset_name = "Fr_to_en_product" #movie ou product
3 english = True

```

```

Entrée [ ]: 1 #Create vectors representing the comments
2 vectorizer = CountVectorizer(lowercase=True, ngram_range=(1, 2), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['review'])
4
5 data_name = dataset_name+"/review+bow+2grams"
6 legend = "No preprocessing. English\nVectorisation with Bow and n-grams=2"

```

Entrée [14]:

```

1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best parameters = {'alpha': 3} grid time : 0.1479

Model Accuracy 1 : 0.5236

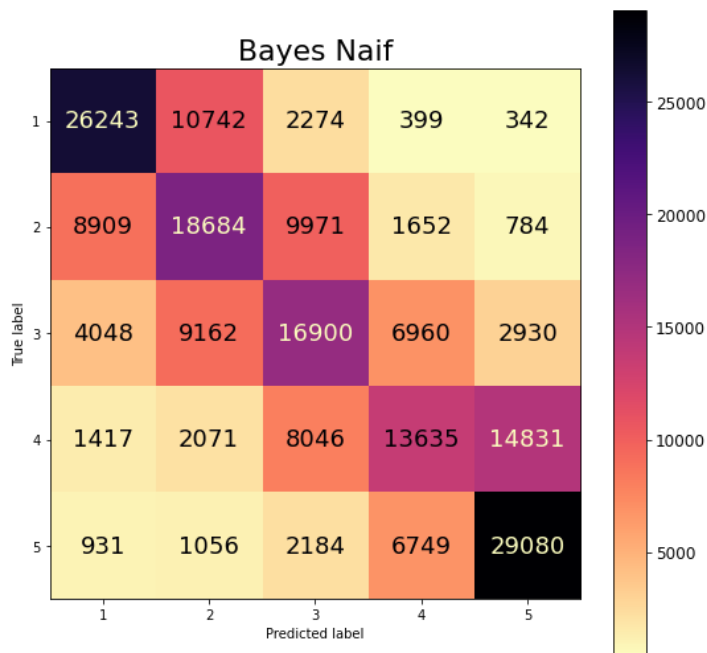
Model Accuracy 2 : 0.522575

Model Accuracy 3 : 0.52175

Model Accuracy 4 : 0.521825

Model Accuracy 5 : 0.5238

fit time : 0.0374



Accuracy = 0.5227

No preprocessing. English

Vectorisation with Bow and n-grams=2

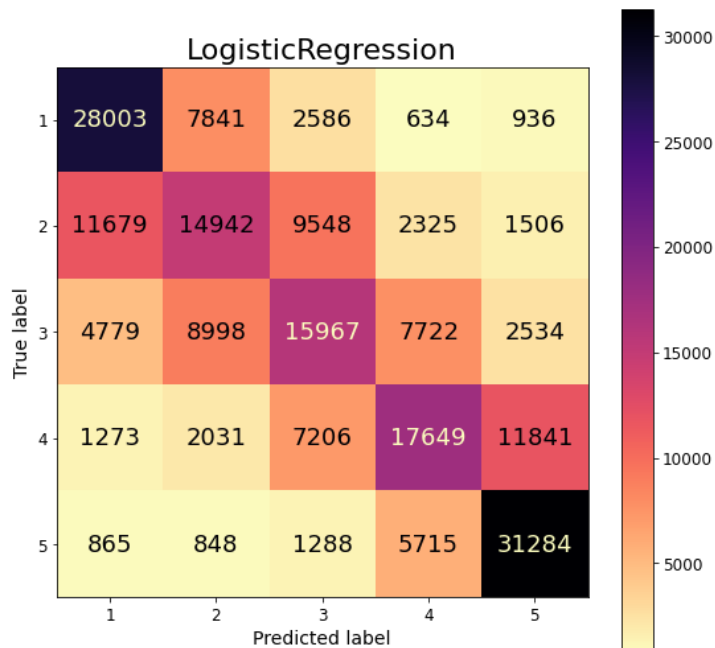
Features = 130229 Elements = 8456987

```

Entrée [172]: 1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits
 Best parameters = {'C': 0.01} grid time : 27.0173
 Model Accuracy 1 : 0.539425
 Model Accuracy 2 : 0.53985
 Model Accuracy 3 : 0.540475
 Model Accuracy 4 : 0.537975
 Model Accuracy 5 : 0.5384
 fit time : 5.1231



Accuracy = 0.5392
 No preprocessing. English
 Vectorisation with Bow and n-grams=3
 Features = 277907 Elements = 12034012

```

Entrée [173]: 1 #Create vectors representing the comments
2 vectorizer = CountVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['review_proc'])
4
5 data_name = dataset_name+"/review_proc+bow+3grams"
6 legend = "Preprocessing. English\nVectorisation with Bow and n-grams=3"

```

Out[173]: <200000x224334 sparse matrix of type '<class 'numpy.int64'>' with 9206079 stored elements in Compressed Sparse Row format>

```

Entrée [174]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best parameters = {'alpha': 2} grid time : 0.2954

Model Accuracy 1 : 0.5297

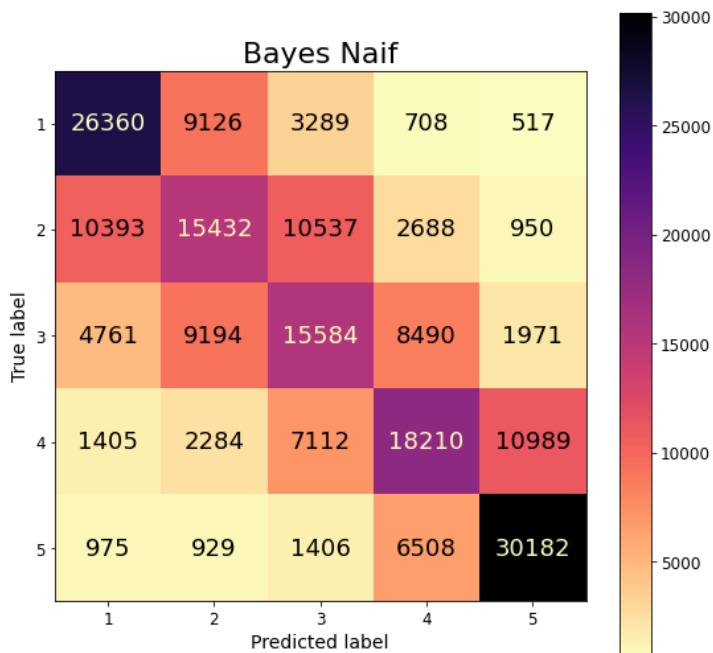
Model Accuracy 2 : 0.529375

Model Accuracy 3 : 0.528525

Model Accuracy 4 : 0.52995

Model Accuracy 5 : 0.52665

fit time : 0.0459



Accuracy = 0.5288

Preprocessing: English

Vectorisation with Bow and n-grams=3

Features = 224334 Elements = 9206079

Entrée [175]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.01} grid time : 7.9634

Model Accuracy 1 : 0.534375

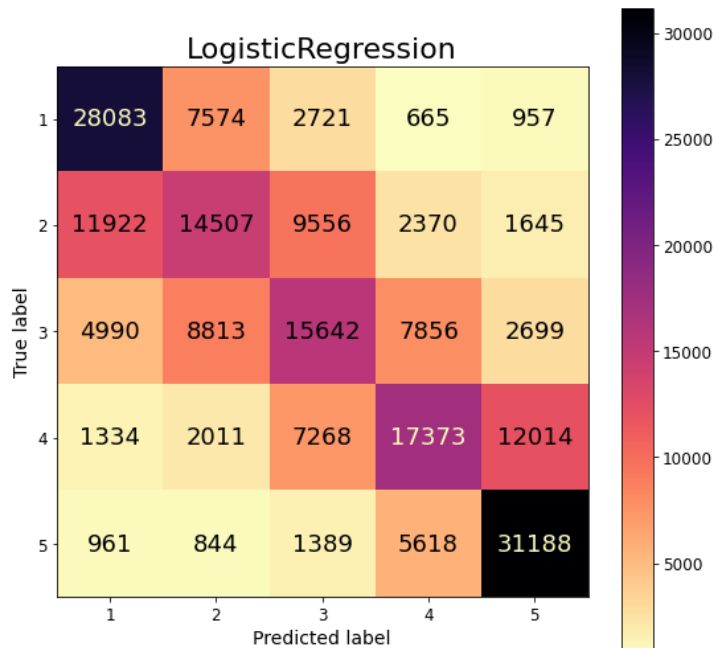
Model Accuracy 2 : 0.533775

Model Accuracy 3 : 0.535675

Model Accuracy 4 : 0.53485

Model Accuracy 5 : 0.53115

fit time : 2.8379



Accuracy = 0.534

Preprocessing. English

Vectorisation with Bow and n-grams=3

Features = 224334 Elements = 9206079

Entrée [176]:

```

1 #Create vectors representing the comments
2 vectorizer = CountVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['review_proc+lemma'])
4
5 data_name = dataset_name+"/review_proc+lemma+bow+3grams"
6 legend = "Preprocessing and lemmatisation. English\nVectorisation with Bow and n-grams=3"

```

Out[176]: <200000x226160 sparse matrix of type '<class 'numpy.int64'>' with 9610435 stored elements in Compressed Sparse Row format>

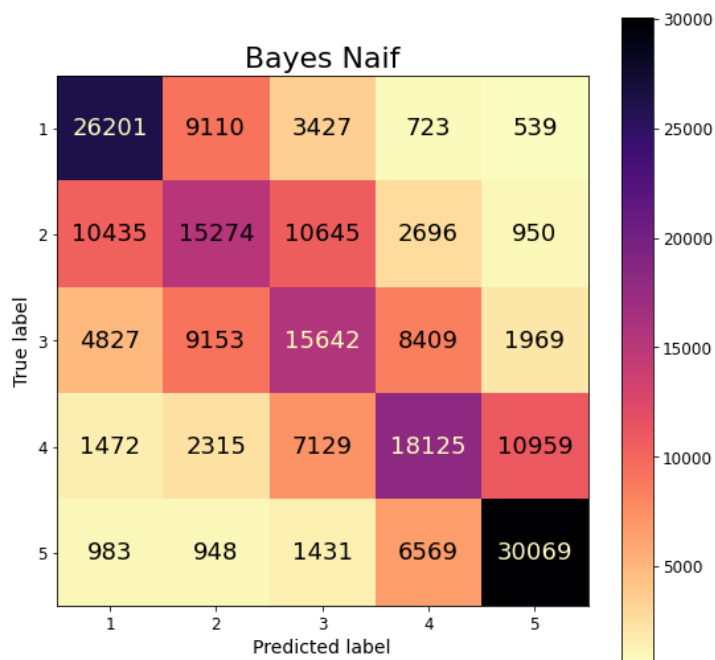
Entrée [177]:

```

1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plo#Create classifier
10 clf = LogisticRegression()
11 name = "LogisticRegression"
12
13 #Run the gridsearch
14 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
15 param0=grid(param_grid0,clf)
16
17 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
18 clf = LogisticRegression(**param0, max_iter = 200)
19 matrix,accuracy = train(clf,bow,Y)
20 plot_matrix(matrix,accuracy,name)t confusion matrix
21 clf = MultinomialNB(**param)
22 matrix,accuracy = train(clf,bow,Y)
23 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
 Best parameters = {'alpha': 2} grid time : 0.2206
 Model Accuracy 1 : 0.526275
 Model Accuracy 2 : 0.526175
 Model Accuracy 3 : 0.526625
 Model Accuracy 4 : 0.5287
 Model Accuracy 5 : 0.525
 fit time : 0.0392



Accuracy = 0.5266
 Preprocessing and lemmatisation. English
 Vectorisation with Bow and n-grams=3
 Features = 226160 Elements = 9610435

Entrée [178]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.01} grid time : 6.4899

Model Accuracy 1 : 0.531225

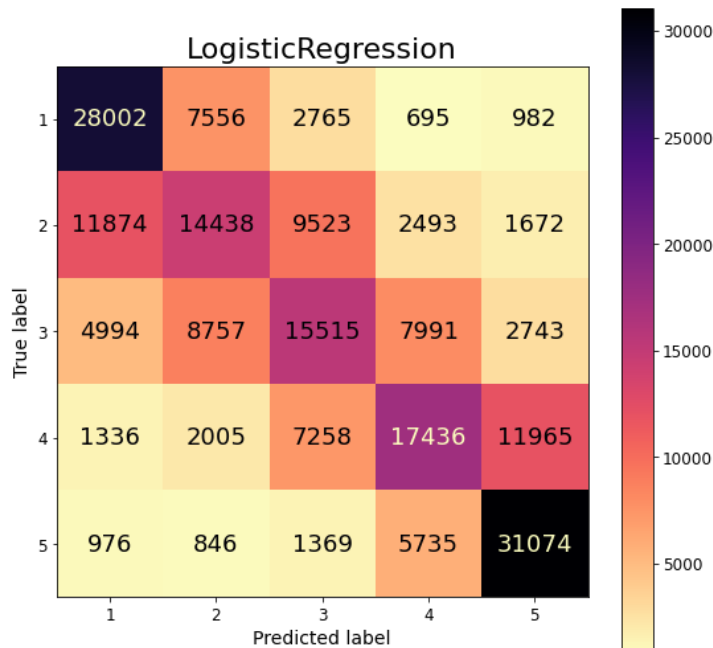
Model Accuracy 2 : 0.5331

Model Accuracy 3 : 0.534725

Model Accuracy 4 : 0.532075

Model Accuracy 5 : 0.5305

fit time : 2.8062



Accuracy = 0.5323

Preprocessing and lemmatisation. English

Vectorisation with Bow and n-grams=3

Features = 226160 Elements = 9610435

Entrée [179]:

```

1 #Create vectors representing the comments
2 if english:
3     load_model = spacy.load('en_core_web_sm') #fr_core_news_sm en_core_web_sm
4 else:
5     load_model = spacy.load('fr_core_news_sm')
6 all_stopwords = load_model.Defaults.stop_words # -stopwords
7
8 vectorizer = CountVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5, stop_words=all_stopwords)
9 bow = vectorizer.fit_transform(data['review_proc+lemma'])
10
11 data_name = dataset_name+"/review_proc+lemma-stopword+bow+3grams"
12 legend = "Preprocessing, lemmatisation and stopwords removal. English\nVectorisation with Bow and n-grams=3"

```

C:\Users\Alexis\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['ll', 've'] not in stop_words.
warnings.warn('Your stop_words may be inconsistent with ')

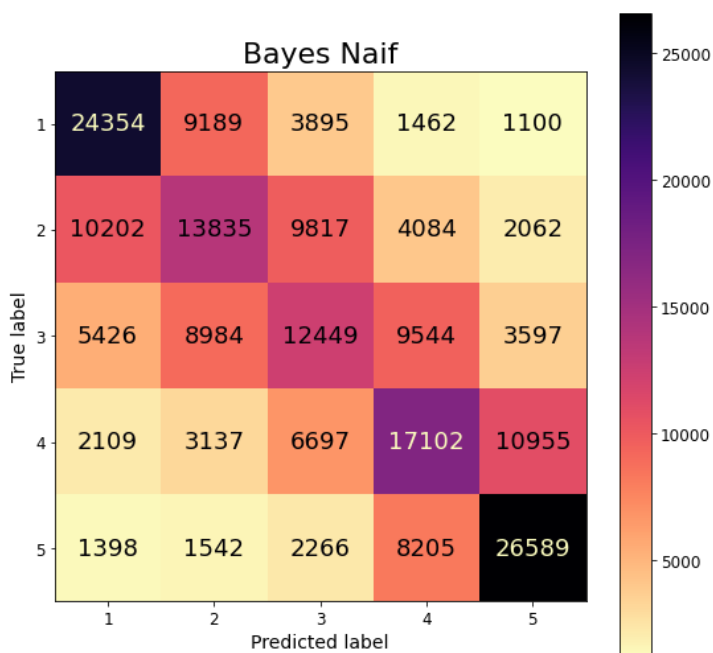
Out[179]: <200000x108114 sparse matrix of type '<class 'numpy.int64'>' with 4128777 stored elements in Compressed Sparse Row format>

```

Entrée [180]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
 Best parameters = {'alpha': 3} grid time : 0.1573
 Model Accuracy 1 : 0.473575
 Model Accuracy 2 : 0.46975
 Model Accuracy 3 : 0.4739
 Model Accuracy 4 : 0.472325
 Model Accuracy 5 : 0.468675
 fit time : 0.0313



Accuracy = 0.4716
 Preprocessing, lemmatisation and stopwords removal. English
 Vectorisation with Bow and n-grams=3
 Features = 108114 Elements = 4128777

```

Entrée [181]: 1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid_search(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.05} grid time : 2.6954

Model Accuracy 1 : 0.4834

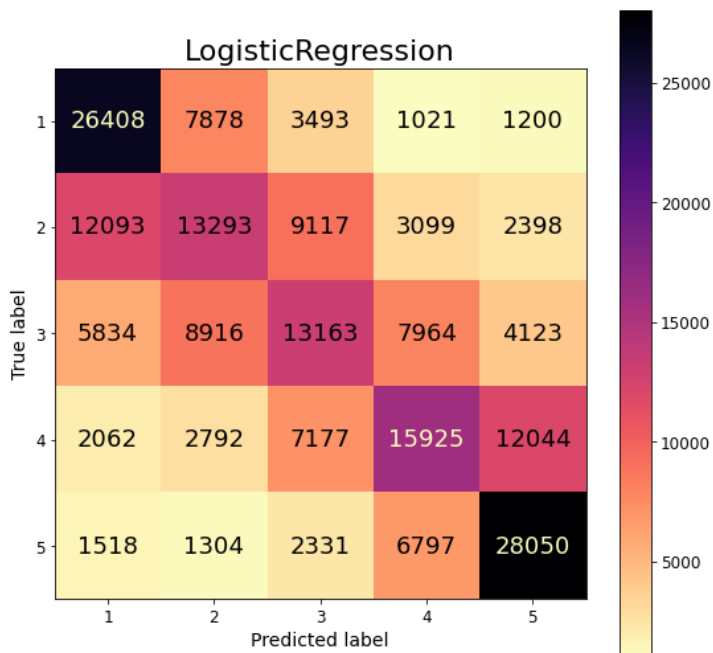
Model Accuracy 2 : 0.484675

Model Accuracy 3 : 0.486175

Model Accuracy 4 : 0.481825

Model Accuracy 5 : 0.4849

fit time : 1.5279



Accuracy = 0.4842

Preprocessing, lemmatisation and stopwords removal. English

Vectorisation with Bow and n-grams=3

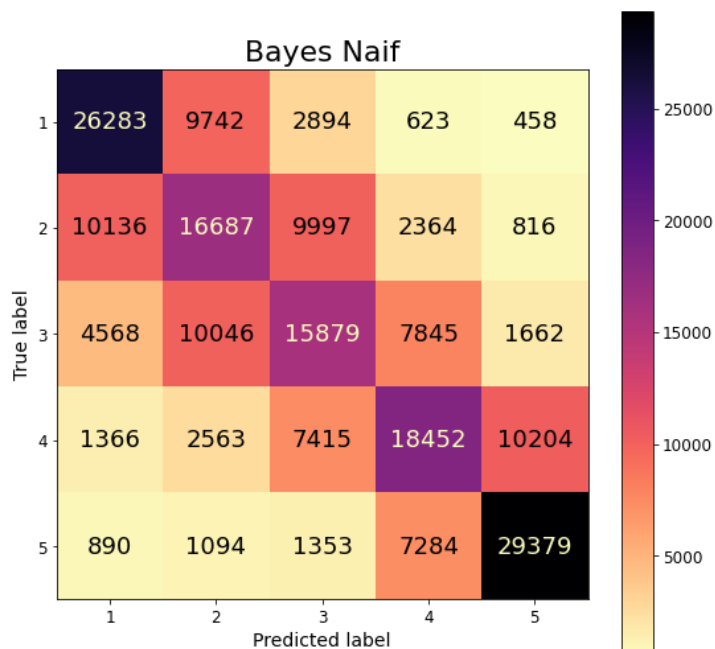
Features = 108114 Elements = 4128777

```
Entrée [182]: 1 #Create vectors representing the comments
2 vectorizer = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['review'])
4
5 data_name = dataset_name+"/review+tfidf+3grams"
6 legend = "No preprocessing. English\nVectorisation with Tfidf and n-grams=3"
```

```
Out[182]: <200000x277907 sparse matrix of type '<class 'numpy.float64'>'
with 12034012 stored elements in Compressed Sparse Row format>
```

```
Entrée [183]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param_grid_search = grid_search(clf, param_grid)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param_grid_search.best_params_)
11 matrix, accuracy = train(clf, bow, Y)
12 plot_matrix(matrix, accuracy, name)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters = {'alpha': 1} grid time : 0.2725
Model Accuracy 1 : 0.53365
Model Accuracy 2 : 0.534425
Model Accuracy 3 : 0.53535
Model Accuracy 4 : 0.5334
Model Accuracy 5 : 0.530175
fit time : 0.0481
```



```
Accuracy = 0.5334
No preprocessing. English
Vectorisation with Tfidf and n-grams=3
Features = 277907 Elements = 12034012
```

Entrée [184]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters = {'C': 0.5} grid time : 21.3852

Model Accuracy 1 : 0.54245

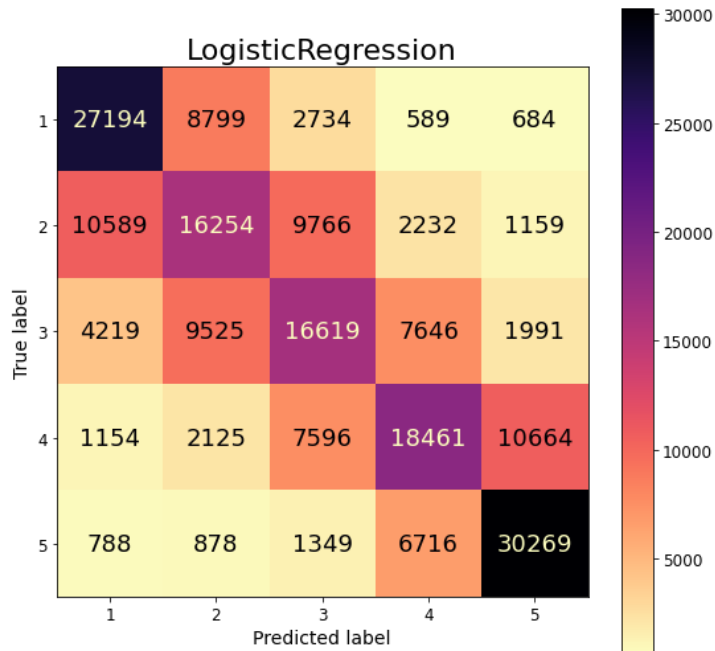
Model Accuracy 2 : 0.54315

Model Accuracy 3 : 0.546725

Model Accuracy 4 : 0.5455

Model Accuracy 5 : 0.5421

fit time : 3.4329



Accuracy = 0.544

No preprocessing. English

Vectorisation with Tfidf and n-grams=3

Features = 277907 Elements = 12034012

Entrée [185]:

```

1 #Create vectors representing the comments
2 vectorizer = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['review_proc'])
4
5 data_name = dataset_name+"/review_proc+tfidf+3grams"
6 legend = "Preprocessing. English\nVectorisation with Tfidf and n-grams=3"

```

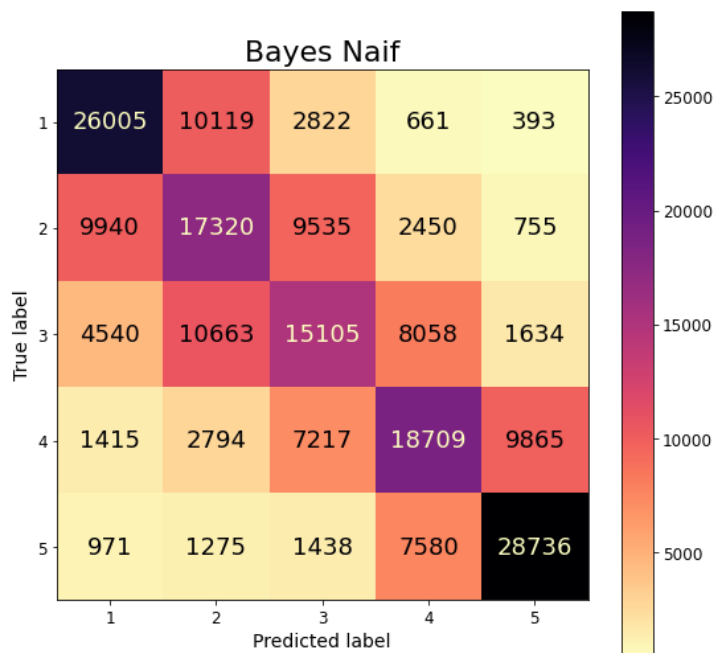
Out[185]: <20000x224334 sparse matrix of type '<class 'numpy.float64'>' with 9206079 stored elements in Compressed Sparse Row format>

```

Entrée [186]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
 Best parameters = {'alpha': 2} grid time : 0.2167
 Model Accuracy 1 : 0.5269
 Model Accuracy 2 : 0.531675
 Model Accuracy 3 : 0.531125
 Model Accuracy 4 : 0.5302
 Model Accuracy 5 : 0.526975
 fit time : 0.0387



Accuracy = 0.5294
 Preprocessing. English
 Vectorisation with Tfidf and n-grams=3
 Features = 224334 Elements = 9206079

Entrée [187]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid_search(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.5} grid time : 6.0099

Model Accuracy 1 : 0.536775

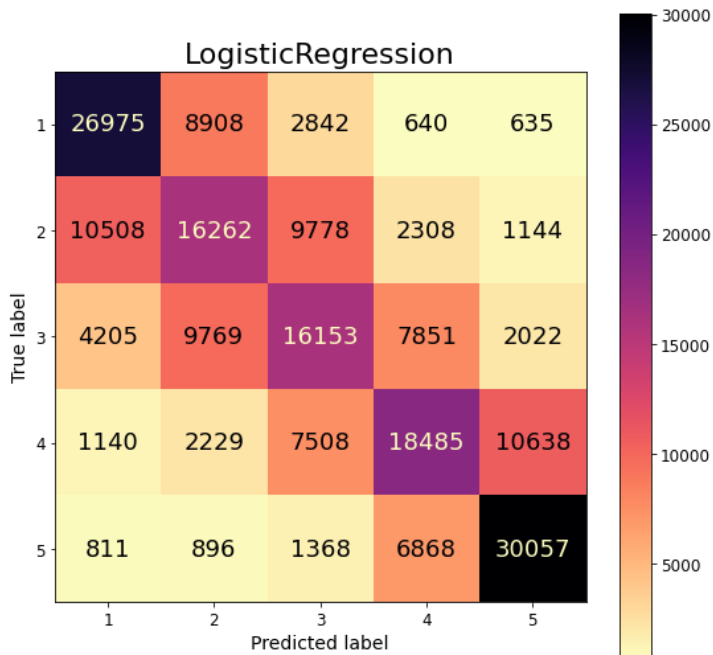
Model Accuracy 2 : 0.541075

Model Accuracy 3 : 0.54175

Model Accuracy 4 : 0.5405

Model Accuracy 5 : 0.5382

fit time : 2.6977



Accuracy = 0.5397

Preprocessing. English

Vectorisation with Tfidf and n-grams=3

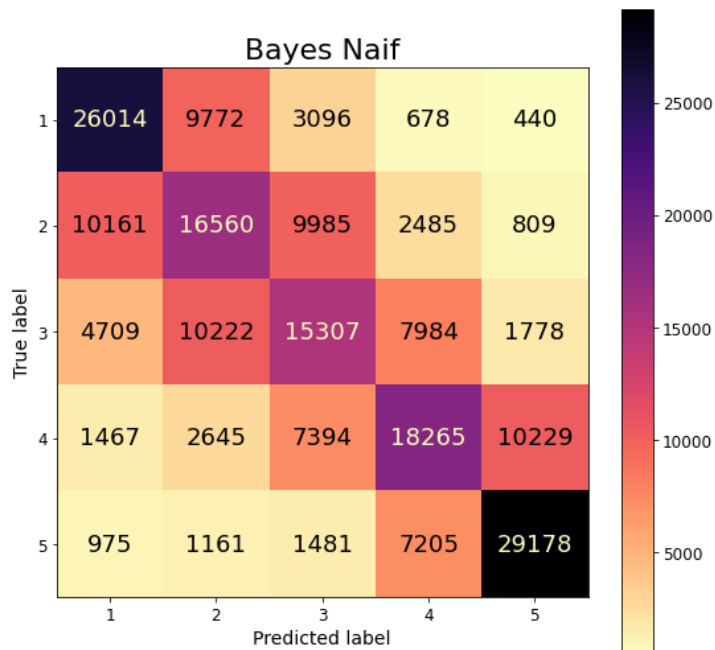
Features = 224334 Elements = 9206079

```
Entrée [188]: 1 #Create vectors representing the comments
2 vectorizer = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['review_proc+lemma'])
4
5 data_name = dataset_name+"/review_proc+lemma+tfidf+3grams"
6 legend = "Preprocessing and lemmatisation. English\nVectorisation with Tfidf and n-grams=3"
```

```
Out[188]: <200000x226160 sparse matrix of type '<class 'numpy.float64'>'
with 9610435 stored elements in Compressed Sparse Row format>
```

```
Entrée [189]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid_search(clf,param_grid)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters = {'alpha': 1} grid time : 0.2244
Model Accuracy 1 : 0.52615
Model Accuracy 2 : 0.527325
Model Accuracy 3 : 0.52725
Model Accuracy 4 : 0.528225
Model Accuracy 5 : 0.52415
fit time : 0.0404
```



```
Accuracy = 0.5266
Preprocessing and lemmatisation. English
Vectorisation with Tfidf and n-grams=3
Features = 226160 Elements = 9610435
```

Entrée [190]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid_search(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.5} grid time : 6.1672

Model Accuracy 1 : 0.53585

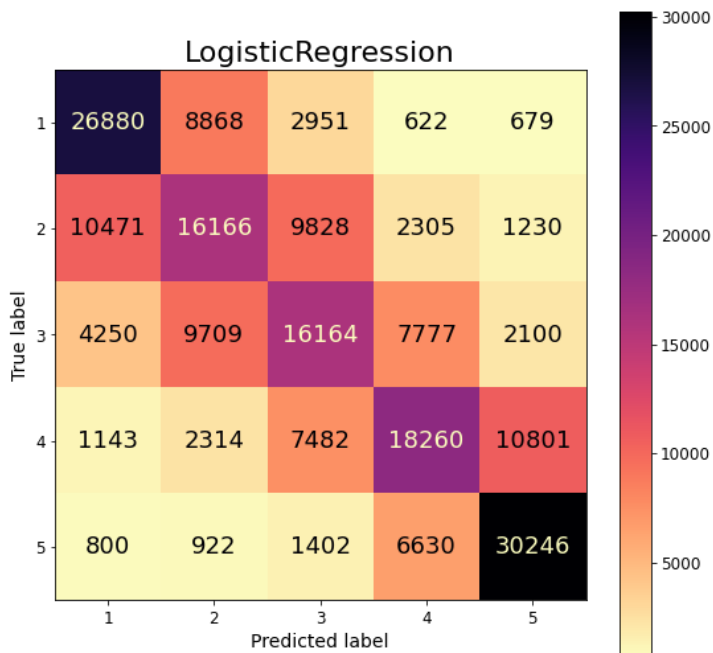
Model Accuracy 2 : 0.539975

Model Accuracy 3 : 0.540675

Model Accuracy 4 : 0.5399

Model Accuracy 5 : 0.5365

fit time : 2.7999



Accuracy = 0.5386

Preprocessing and lemmatisation. English

Vectorisation with Tfidf and n-grams=3

Features = 226160 Elements = 9610435

Entrée [191]:

```

1 #Create vectors representing the comments
2 if english:
3     load_model = spacy.load('en_core_web_sm') #fr_core_news_sm en_core_web_sm
4 else:
5     load_model = spacy.load('fr_core_news_sm')
6 all_stopwords = load_model.Defaults.stop_words # -stopwords
7
8 vectorizer = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5, stop_words=all_stopwords)
9 bow = vectorizer.fit_transform(data['review_proc+lemma'])
10
11 data_name = dataset_name+"/review_proc+lemma-stopword+tfidf+3grams"
12 legend = "Preprocessing, lemmatisation and stopwords removal. English\nVectorisation with Tfidf and n-grams=3"

```

C:\Users\Alexis\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['ll', 've'] not in stop_words.
warnings.warn('Your stop_words may be inconsistent with ')

Out[191]: <20000x108114 sparse matrix of type '<class 'numpy.float64''>
with 4128777 stored elements in Compressed Sparse Row format>

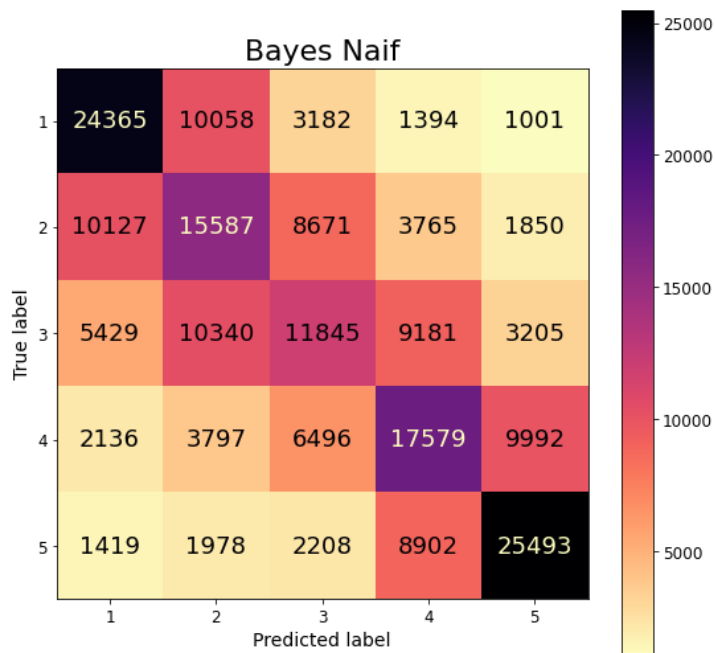
Entrée [192]:

```

1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters = {'alpha': 7} grid time : 0.1673
Model Accuracy 1 : 0.474
Model Accuracy 2 : 0.473825
Model Accuracy 3 : 0.477925
Model Accuracy 4 : 0.475975
Model Accuracy 5 : 0.47
fit time : 0.0334



Accuracy = 0.4743
Preprocessing, lemmatisation and stopwords removal. English
Vectorisation with Tfidf and n-grams=3
Features = 108114 Elements = 4128777

```

Entrée [193]: 1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid_search(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.5} grid time : 2.6578

Model Accuracy 1 : 0.486475

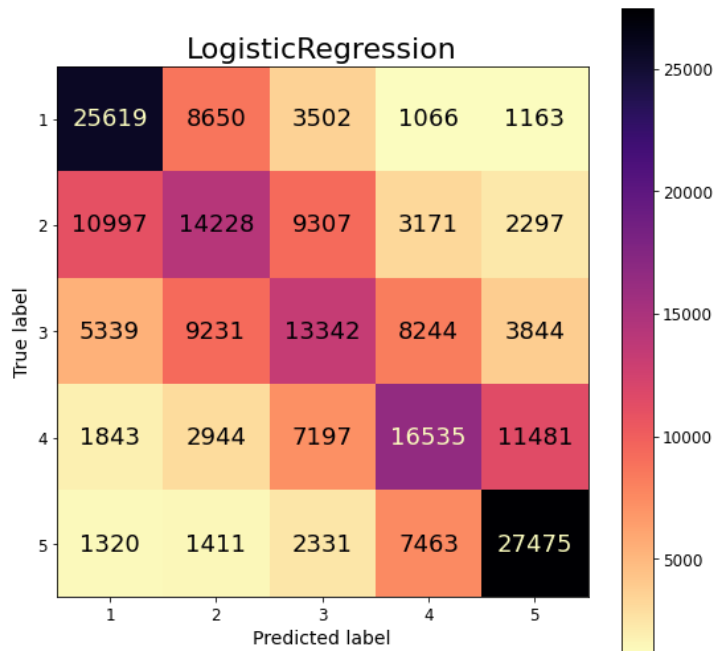
Model Accuracy 2 : 0.484525

Model Accuracy 3 : 0.489125

Model Accuracy 4 : 0.48375

Model Accuracy 5 : 0.4861

fit time : 1.3071



Accuracy = 0.486

Preprocessing, lemmatisation and stopwords removal. English

Vectorisation with Tfidf and n-grams=3

Features = 108114 Elements = 4128777

Traduction

```

Entrée [ ]: 1 #Redefine Language
2 english = False

```

```

Entrée [11]: 1 #Create vectors representing the comments
2 vectorizer = CountVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['translation'])
4
5 data_name = dataset_name+"/translation+bow+3grams"
6 legend = "No preprocessing. French Translated\nVectorisation with Bow and n-grams=3"

```

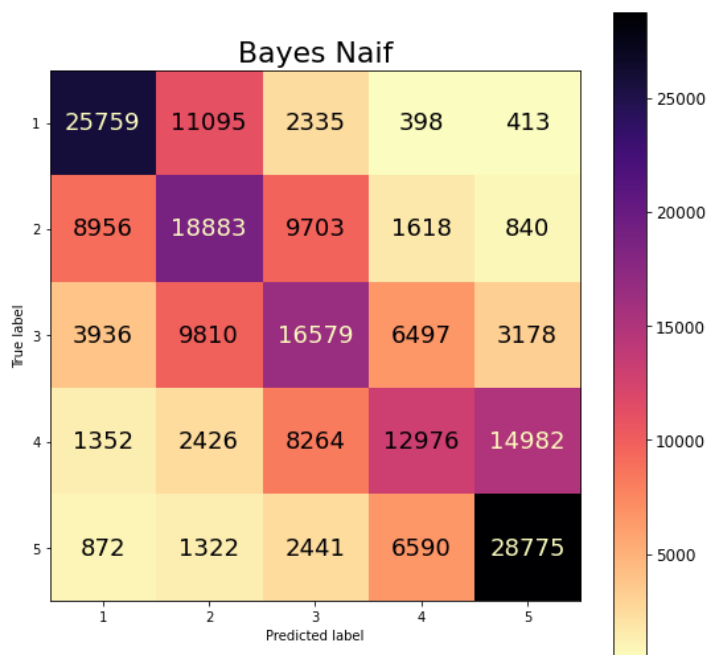
Out[11]: <200000x206700 sparse matrix of type '<class 'numpy.int64'>' with 9413993 stored elements in Compressed Sparse Row format>

```

Entrée [12]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
 Best parameters = {'alpha': 3} grid time : 0.1741
 Model Accuracy 1 : 0.516925
 Model Accuracy 2 : 0.5137
 Model Accuracy 3 : 0.5161
 Model Accuracy 4 : 0.513025
 Model Accuracy 5 : 0.51455
 fit time : 0.0402



Accuracy = 0.5149
 No preprocessing. French Translated
 Vectorisation with Bow and n-grams=3
 Features = 206700 Elements = 9413993

Entrée [13]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters = {'C': 0.01} grid time : 12.2953

Model Accuracy 1 : 0.530875

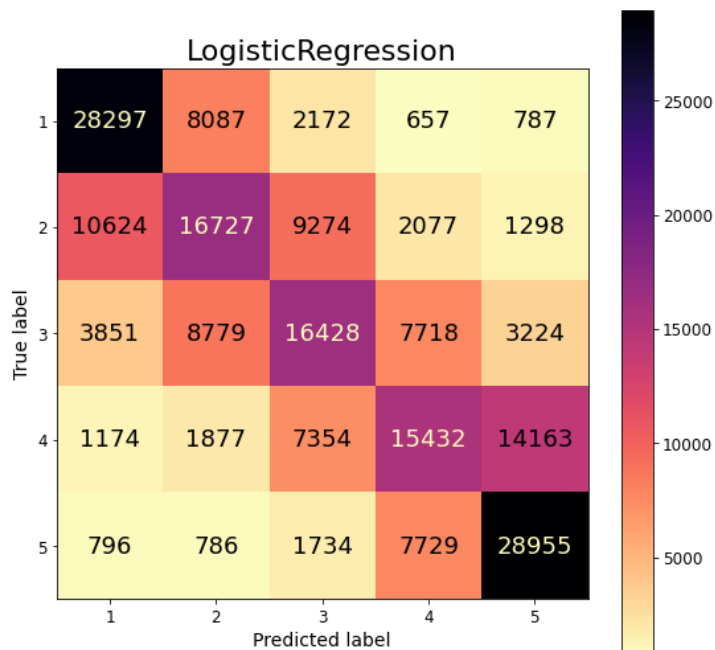
Model Accuracy 2 : 0.5285

Model Accuracy 3 : 0.528175

Model Accuracy 4 : 0.527925

Model Accuracy 5 : 0.5305

fit time : 5.028



Accuracy = 0.5292

No preprocessing. French Translated

Vectorisation with Bow and n-grams=3

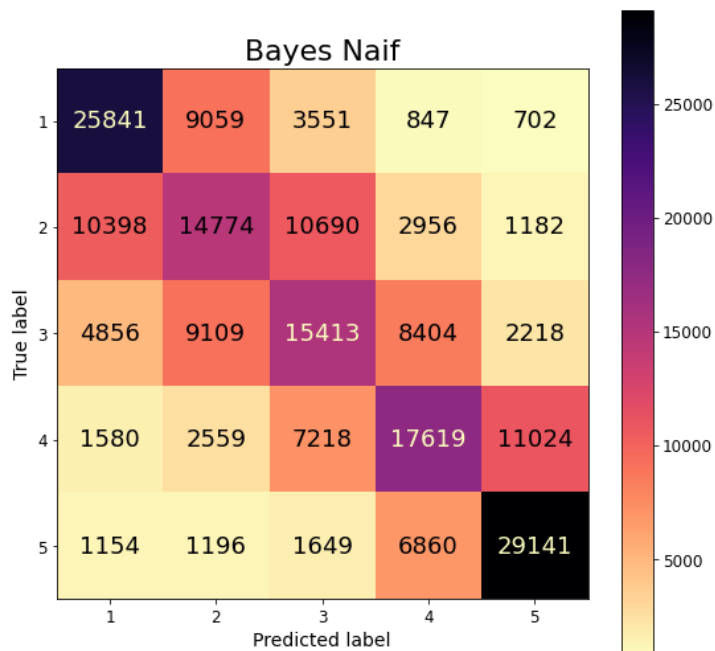
Features = 206700 Elements = 9413993

```
Entrée [198]: 1 #Create vectors representing the comments
2 vectorizer = CountVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['translation_proc'])
4
5 data_name = dataset_name+"/translation_proc+bow+3grams"
6 legend = "Preprocessing. French Translated\nVectorisation with Bow and n-grams=3"
```

```
Out[198]: <200000x218621 sparse matrix of type '<class 'numpy.int64'>'
with 9076900 stored elements in Compressed Sparse Row format>
```

```
Entrée [199]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters = {'alpha': 2} grid time : 0.259
Model Accuracy 1 : 0.51495
Model Accuracy 2 : 0.513075
Model Accuracy 3 : 0.516175
Model Accuracy 4 : 0.513425
Model Accuracy 5 : 0.512075
fit time : 0.0418
```



```
Accuracy = 0.5139
Preprocessing. French Translated
Vectorisation with Bow and n-grams=3
Features = 218621 Elements = 9076900
```

Entrée [200]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid_search(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.05} grid time : 6.3238

Model Accuracy 1 : 0.51835

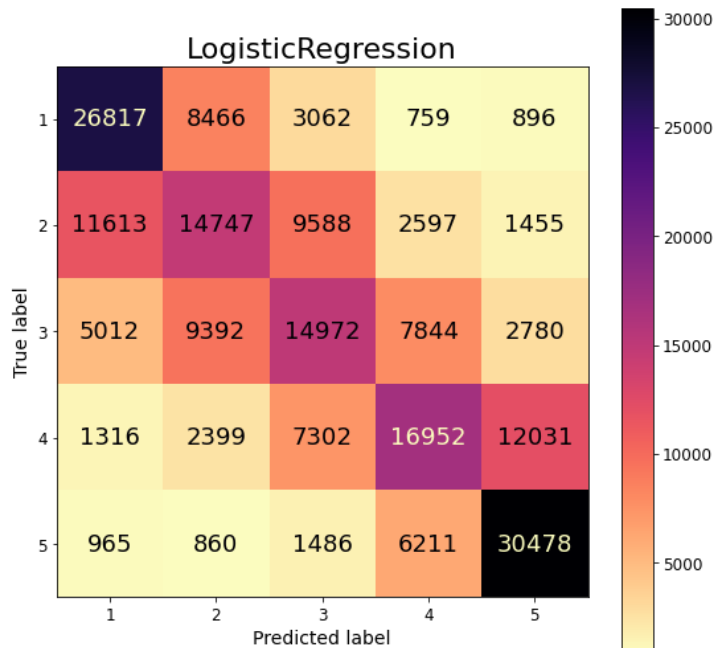
Model Accuracy 2 : 0.51935

Model Accuracy 3 : 0.52325

Model Accuracy 4 : 0.51915

Model Accuracy 5 : 0.51905

fit time : 2.7047



Accuracy = 0.5198

Preprocessing. French Translated

Vectorisation with Bow and n-grams=3

Features = 218621 Elements = 9076900

Entrée [201]:

```

1 #Create vectors representing the comments
2 vectorizer = CountVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['translation_proc+lemma'])
4
5 data_name = dataset_name+"/translation_proc+lemma+bow+3grams"
6 legend = "Preprocessing and lemmatisation. French Translated\nVectorisation with Bow and n-grams=3"

```

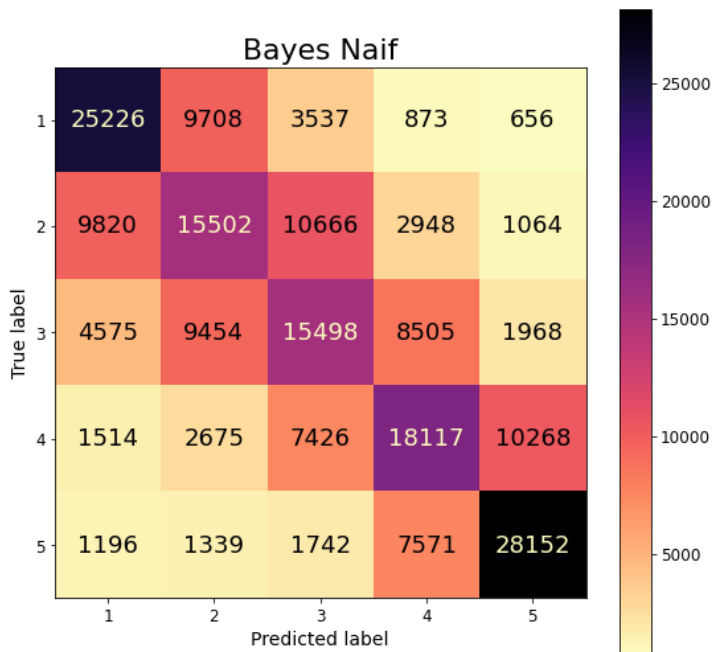
Out[201]: <200000x216200 sparse matrix of type '<class 'numpy.int64'>' with 9347525 stored elements in Compressed Sparse Row format>

```

Entrée [202]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
 Best parameters = {'alpha': 3} grid time : 0.2209
 Model Accuracy 1 : 0.5131
 Model Accuracy 2 : 0.5106
 Model Accuracy 3 : 0.514625
 Model Accuracy 4 : 0.512875
 Model Accuracy 5 : 0.511175
 fit time : 0.0392



Accuracy = 0.5125
 Preprocessing and lemmatisation. French Translated
 Vectorisation with Bow and n-grams=3
 Features = 216200 Elements = 9347525

Entrée [203]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.01} grid time : 6.5302

Model Accuracy 1 : 0.519875

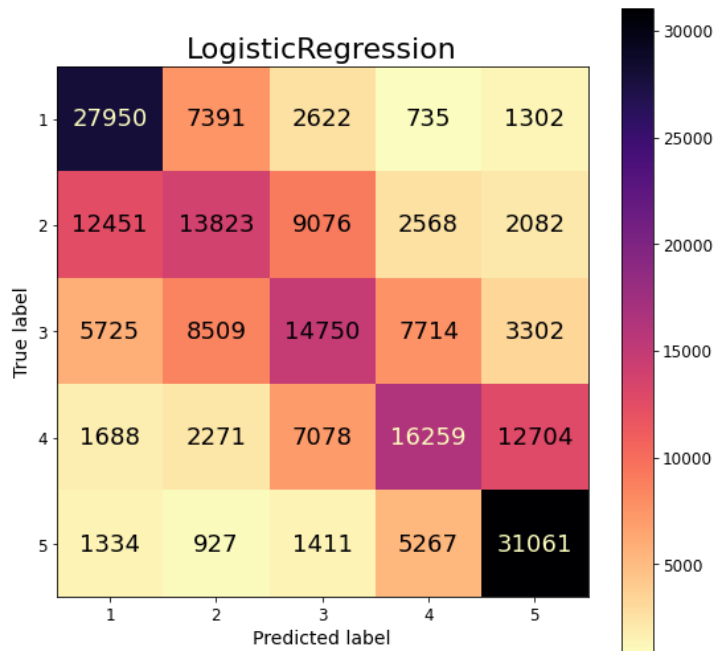
Model Accuracy 2 : 0.519125

Model Accuracy 3 : 0.520675

Model Accuracy 4 : 0.519125

Model Accuracy 5 : 0.517275

fit time : 2.7418



Accuracy = 0.5192

Preprocessing and lemmatisation. French Translated

Vectorisation with Bow and n-grams=3

Features = 216200 Elements = 9347525

Entrée [56]:

```

1 #Create vectors representing the comments
2 if english:
3     load_model = spacy.load('en_core_web_sm') #fr_core_news_sm en_core_web_sm
4 else:
5     load_model = spacy.load('fr_core_news_sm')
6 all_stopwords = load_model.Defaults.stop_words # -stopwords
7
8 vectorizer = CountVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5, stop_words=all_stopwords)
9 bow = vectorizer.fit_transform(data['translation_proc+lemma'])
10
11 data_name = dataset_name+"/translation_proc+lemma-stopword+bow+3grams"
12 legend = "Preprocessing, lemmatisation and stopwords removal. French Translated\nVectorisation with Bow and n-grams=3"

```

C:\Users\Alexis\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['ll', 've'] not in stop_words.
warnings.warn('Your stop_words may be inconsistent with '

Out[56]: <200000x79668 sparse matrix of type '<class 'numpy.int64'>' with 3197732 stored elements in Compressed Sparse Row format>

Entrée [57]:

```

1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best parameters = {'alpha': 5} grid time : 0.1216

Model Accuracy 1 : 0.469775

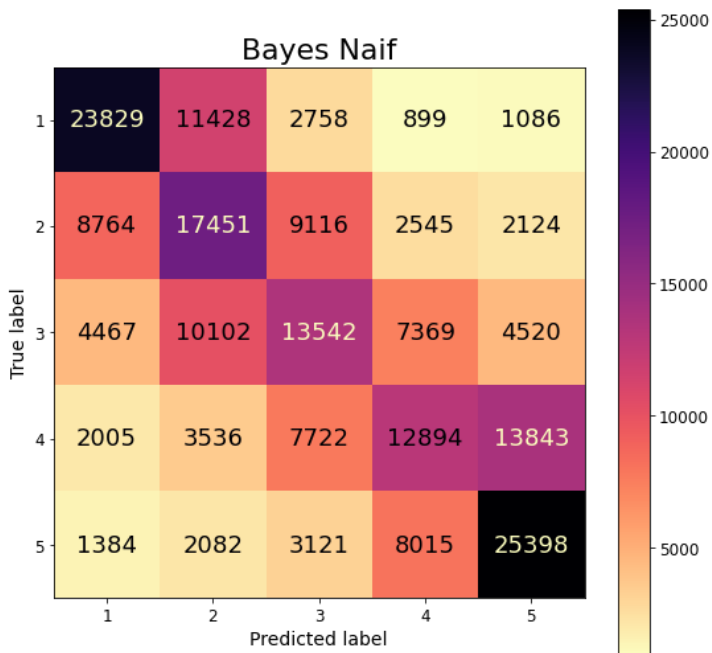
Model Accuracy 2 : 0.4637

Model Accuracy 3 : 0.465525

Model Accuracy 4 : 0.4628

Model Accuracy 5 : 0.46605

fit time : 0.0311



Accuracy = 0.4656

Preprocessing, lemmatisation and stopwords removal. English Translated

Vectorisation with Bow and n-grams=3

Features = 79668 Elements = 3197732

Entrée [58]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters = {'C': 0.05} grid time : 3.9618

Model Accuracy 1 : 0.479125

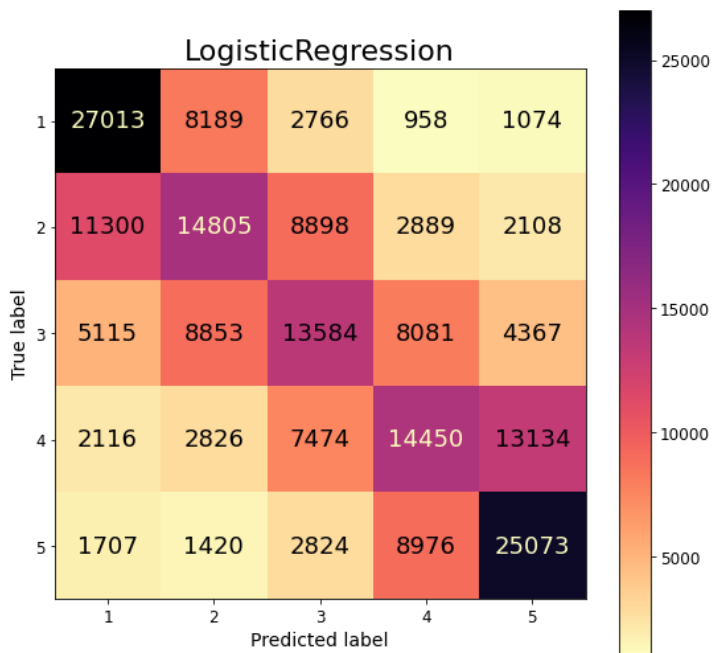
Model Accuracy 2 : 0.472725

Model Accuracy 3 : 0.4721

Model Accuracy 4 : 0.4702

Model Accuracy 5 : 0.478975

fit time : 1.0436



Accuracy = 0.4746

Preprocessing, lemmatisation and stopwords removal. English Translated

Vectorisation with Bow and n-grams=3

Features = 79668 Elements = 3197732

```

Entrée [10]: 1 #Create vectors representing the comments
2 vectorizer = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['translation'])
4
5 data_name = dataset_name+"/translation+tfidf+3grams"
6 legend = "No preprocessing. French Translated\nVectorisation with Tfidf and n-grams=3"

```

```

Out[10]: <200000x303766 sparse matrix of type '<class 'numpy.float64'>'
        with 13851754 stored elements in Compressed Sparse Row format>

```

```

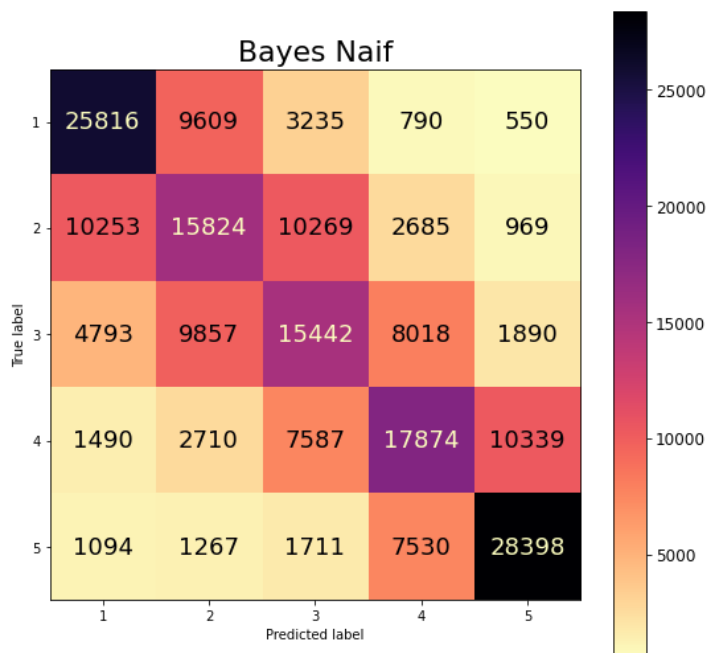
Entrée [11]: 1 #Create classifier
2 name = "Bayes Naïf"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param_grid_search = GridSearchCV(clf, param_grid)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param_grid_search.best_params_)
11 matrix, accuracy = train(clf, bow, Y)
12 plot_matrix(matrix, accuracy, name)

```

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters = {'alpha': 0.7} grid time : 0.3254
Model Accuracy 1 : 0.5162
Model Accuracy 2 : 0.517575
Model Accuracy 3 : 0.51955
Model Accuracy 4 : 0.515125
Model Accuracy 5 : 0.5154
fit time : 0.0552

```



```

Accuracy = 0.5168
No preprocessing. French Translated
Vectorisation with Tfidf and n-grams=3
Features = 303766 Elements = 13851754

```

Entrée [12]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters = {'C': 0.5} grid time : 20.5236

Model Accuracy 1 : 0.53

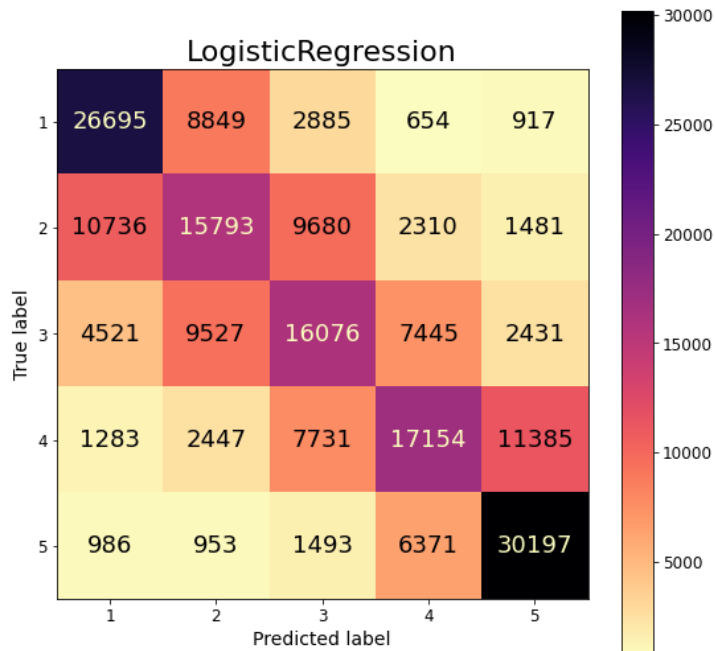
Model Accuracy 2 : 0.52865

Model Accuracy 3 : 0.531425

Model Accuracy 4 : 0.5309

Model Accuracy 5 : 0.5269

fit time : 4.0947



Accuracy = 0.5296

No preprocessing. French Translated

Vectorisation with Tfidf and n-grams=3

Features = 303766 Elements = 13851754

Entrée [13]:

```

1 #Create vectors representing the comments
2 vectorizer = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['translation_proc'])
4
5 data_name = dataset_name+"/translation_proc+tfidf+3grams"
6 legend = "Preprocessing. French Translated\nVectorisation with Tfidf and n-grams=3"

```

Out[13]: <200000x218621 sparse matrix of type '<class 'numpy.float64'>' with 9076900 stored elements in Compressed Sparse Row format>

```

Entrée [14]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best parameters = {'alpha': 1} grid time : 0.2608

Model Accuracy 1 : 0.511475

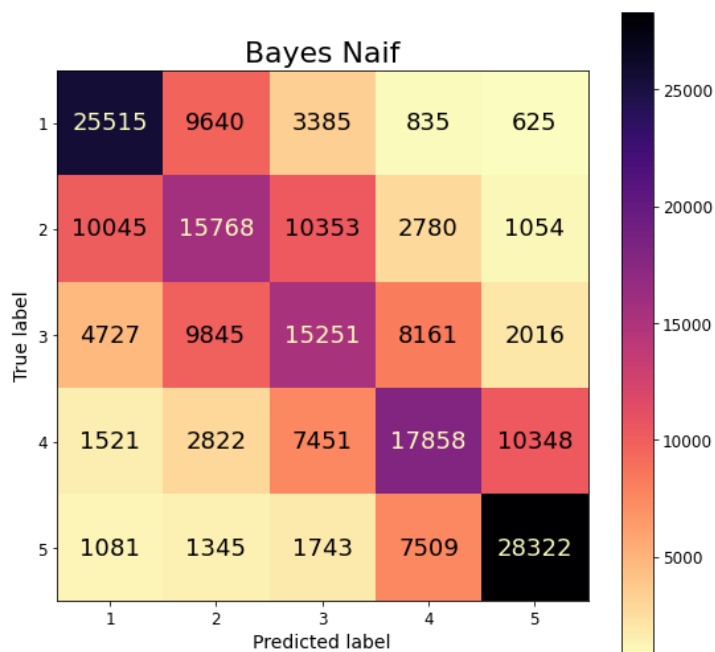
Model Accuracy 2 : 0.511275

Model Accuracy 3 : 0.517775

Model Accuracy 4 : 0.513525

Model Accuracy 5 : 0.5138

fit time : 0.0413



Accuracy = 0.5136

Preprocessing. French Translated

Vectorisation with Tfidf and n-grams=3

Features = 218621 Elements = 9076900

Entrée [15]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.5} grid time : 5.7917

Model Accuracy 1 : 0.52715

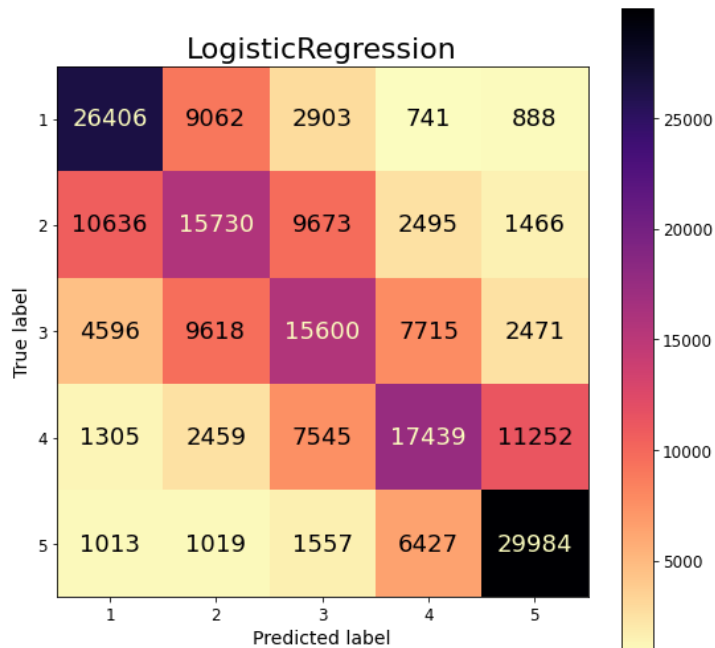
Model Accuracy 2 : 0.523575

Model Accuracy 3 : 0.529025

Model Accuracy 4 : 0.5247

Model Accuracy 5 : 0.524525

fit time : 2.446



Accuracy = 0.5258

Preprocessing. French Translated

Vectorisation with Tfidf and n-grams=3

Features = 218621 Elements = 9076900

Entrée [16]:

```

1 #Create vectors representing the comments
2 vectorizer = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5)
3 bow = vectorizer.fit_transform(data['translation_proc+lemma'])
4
5 data_name = dataset_name+"/translation_proc+lemma+tfidf+3grams"
6 legend = "Preprocessing and lemmatisation. French Translated\nVectorisation with Tfidf and n-grams=3"

```

Out[16]: <200000x216200 sparse matrix of type '<class 'numpy.float64'>' with 9347525 stored elements in Compressed Sparse Row format>

Entrée [17]:

```

1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best parameters = {'alpha': 1} grid time : 0.2167

Model Accuracy 1 : 0.51245

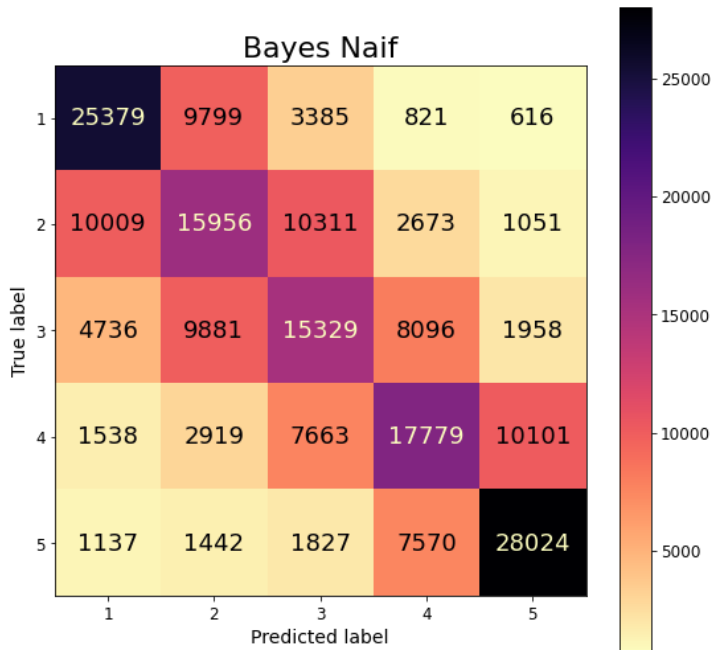
Model Accuracy 2 : 0.5093

Model Accuracy 3 : 0.515675

Model Accuracy 4 : 0.513275

Model Accuracy 5 : 0.510975

fit time : 0.039



Accuracy = 0.5123

Preprocessing and lemmatisation. French Translated

Vectorisation with Tfidf and n-grams=3

Features = 216200 Elements = 9347525

Entrée [18]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters = {'C': 0.5} grid time : 5.5767

Model Accuracy 1 : 0.5246

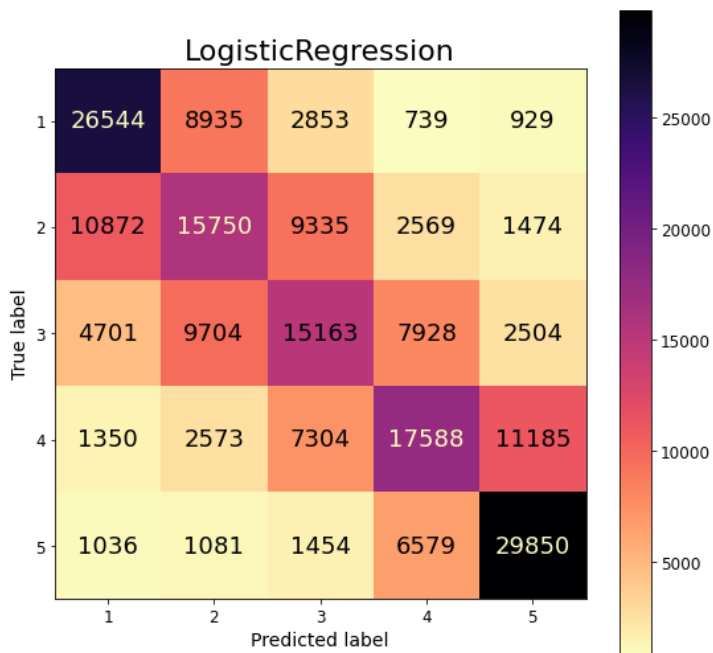
Model Accuracy 2 : 0.5223

Model Accuracy 3 : 0.5295

Model Accuracy 4 : 0.524575

Model Accuracy 5 : 0.5214

fit time : 2.4331



Accuracy = 0.5245

Preprocessing and lemmatisation. French Translated

Vectorisation with Tfidf and n-grams=3

Features = 216200 Elements = 9347525

Entrée [59]:

```

1 #Create vectors representing the comments
2 if english:
3     load_model = spacy.load('en_core_web_sm') #fr_core_news_sm en_core_web_sm
4 else:
5     load_model = spacy.load('fr_core_news_sm')
6 all_stopwords = load_model.Defaults.stop_words # -stopwords
7
8 vectorizer = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), max_df=0.90, min_df=5, stop_words=all_stopwords)
9 bow = vectorizer.fit_transform(data['translation_proc+lemma'])
10
11 data_name = dataset_name+"/translation_proc+lemma-stopword+tfidf+3grams"
12 legend = "Preprocessing, lemmatisation and stopwords removal. French Translated\nVectorisation with Tfidf and n-grams=3"

```

C:\Users\Alexis\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['ll', 've'] not in stop_words.
warnings.warn('Your stop_words may be inconsistent with ')

Out[59]: <200000x79668 sparse matrix of type '<class 'numpy.float64'>' with 3197732 stored elements in Compressed Sparse Row format>

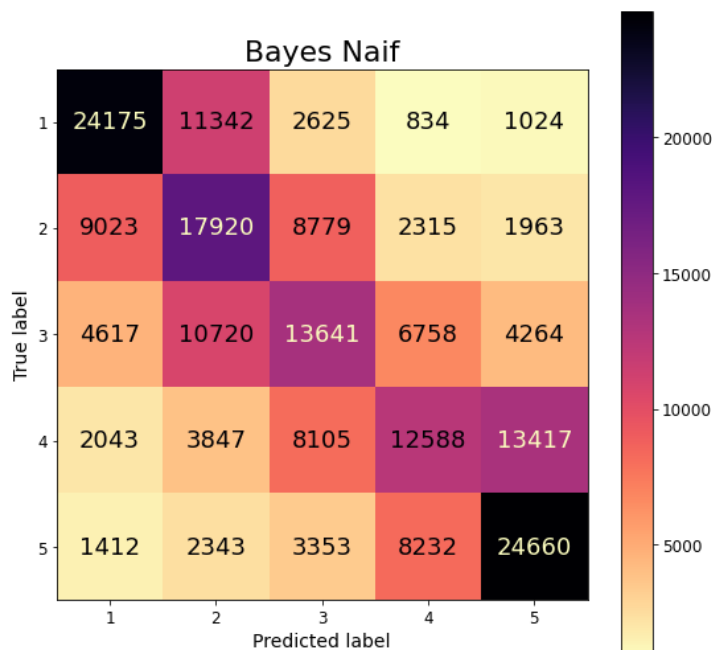
Entrée [60]:

```

1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters = {'alpha': 5} grid time : 0.0954
Model Accuracy 1 : 0.468625
Model Accuracy 2 : 0.462225
Model Accuracy 3 : 0.4652
Model Accuracy 4 : 0.4621
Model Accuracy 5 : 0.46645
fit time : 0.0317



Accuracy = 0.4649
Preprocessing, lemmatisation and stopwords removal. English Translated
Vectorisation with Tfidf and n-grams=3
Features = 79668 Elements = 3197732

Entrée [61]:

```

1 #Create classifier
2 clf = LogisticRegression()
3 name = "LogisticRegression"
4
5 #Run the gridsearch
6 param_grid0 = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param0=grid(param_grid0,clf)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param0, max_iter = 200)
11 matrix,accuracy = train(clf,bow,Y)
12 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters = {'C': 0.5} grid time : 3.6112

Model Accuracy 1 : 0.48175

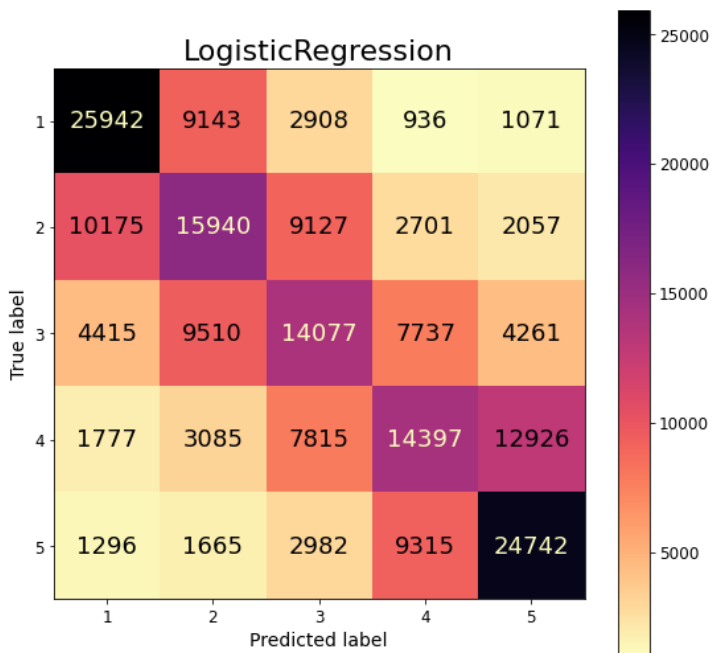
Model Accuracy 2 : 0.474275

Model Accuracy 3 : 0.47445

Model Accuracy 4 : 0.4692

Model Accuracy 5 : 0.47775

fit time : 1.1197



Accuracy = 0.4755

Preprocessing, lemmatisation and stopwords removal. English Translated

Vectorisation with Tfidf and n-grams=3

Features = 79668 Elements = 3197732

5. Performing Sentiment Analysis using Text Classification (Step 2 & 3)

Dependencies

```
Entrée [1]: 1 import pandas as pd
2 import numpy as np
3 #pd.set_option('display.max_rows', 50)
4 #pd.set_option('display.max_colwidth', 1000)
5 from sklearn.feature_extraction.text import CountVectorizer
6 from sklearn.feature_extraction.text import TfidfVectorizer
7 from nltk.tokenize import RegexpTokenizer
8 from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, cross_validate
9 from sklearn.naive_bayes import MultinomialNB
10 from sklearn.linear_model import LogisticRegression
11 from sklearn.pipeline import Pipeline
12 from sklearn import metrics
13 import pickle
14 import spacy
15 import re
16 import seaborn as sns
17 import matplotlib.pyplot as plt
18 import time
19 import warnings
20 from sklearn import exceptions
21 warnings.filterwarnings("ignore", category=exceptions.ConvergenceWarning)
```

Functions

```
Entrée [2]: 1 #Takes the choosen values for hyperameter and a classifier as parameters
2 #and perform a gridsearch with a scoring based on accuracy
3 def grid(param_grid,model,bow,Y):
4     start = time.time()
5     grid_clf = GridSearchCV(model, param_grid = param_grid, scoring = 'accuracy', n_jobs=2,verbose=4, cv=cv)
6     grid_clf.fit(bow, y=Y) #ou sur train
7     end = time.time()
8     print("Best parameters = ", grid_clf.best_params_, " grid time : ", round((end - start)/60, 4))
9     return(grid_clf.best_params_)
```

```
Entrée [3]: 1 #Takes features and the target from 2 datasets with a pipeline as parameters
2 #and perform the split, training and evaluation (step 2)
3 def train (pipeline,X0,Y0,X1,Y1):
4     start = time.time()
5     matrix = np.zeros((n,n))
6     accuracy = 0
7     run = 0
8     for train_ix, test_ix in CV.split(X0, Y0):
9         run+=1
10        xtrain, xtest = X0[train_ix], X1[test_ix]
11        ytrain, ytest = Y0[train_ix], Y1[test_ix]
12
13        #Training
14        pipeline.fit(xtrain,ytrain)
15
16        # Predicting with a test set
17        predicted = pipeline.predict(xtest)
18
19        # Accuracy
20        a = metrics.accuracy_score(ytest, predicted)
21        print("Model Accuracy ",run," : ", a)
22        accuracy = accuracy + a
23
24        #Confusion matrix
25        m = metrics.confusion_matrix(ytest, predicted)
26        matrix = matrix + m
27
28    end = time.time()
29    print("fit time : ", round((end - start)/60, 4))
30    return matrix,accuracy/cv
```

```

Entrée [165]: 1 #Takes features and the target from 3 datasets with a pipeline as parameters and perform the split, training and evaluation
2 def train_cross (pipeline,X0,Y0,X1,Y1,X2,Y2):
3     start = time.time()
4     matrix1 = np.zeros((n,n))
5     matrix2 = np.zeros((n,n))
6     accuracy1 = 0
7     accuracy2 = 0
8     run = 0
9     for train_ix, test_ix in CV.split(X0, Y0):
10         run+=1
11         xtrain, x1test, x2test = X0[train_ix], X1[test_ix], X2[test_ix] #remplacer X1 et Y1 par X0 et Y0 pour tout faire
12         ytrain, y1test, y2test = Y0[train_ix], Y1[test_ix], Y2[test_ix]
13
14         #Training
15         pipeline.fit(xtrain,ytrain)
16
17         # Predicting with a test set
18         predicted1 = pipeline.predict(x1test)
19         predicted2 = pipeline.predict(x2test)
20
21         # Accuracy
22         a1 = metrics.accuracy_score(y1test, predicted1)
23         print("Model Accuracy ",run," : ", a1)
24         accuracy1 = accuracy1 + a1
25
26         a2 = metrics.accuracy_score(y2test, predicted2)
27         print("Model Accuracy ",run," : ", a2)
28         accuracy2 = accuracy2 + a2
29
30         #Confusion matrix
31         m1 = metrics.confusion_matrix(y1test, predicted1)
32         matrix1 = matrix1 + m1
33
34         m2 = metrics.confusion_matrix(y2test, predicted2)
35         matrix2 = matrix2 + m2
36
37     end = time.time()
38     print("fit time : ", round((end - start)/60, 4))
39     return matrix1,accuracy1,cv,matrix2,accuracy2/cv
40

```

```

Entrée [4]: 1 #Takes the confusion matrix, accuracy and a name as parameters and plot the matrix with all information
2 def plot_matrix (matrix,accuracy,name):
3     fig, ax = plt.subplots()
4     fig.set_size_inches(9, 9)
5
6     if n == 2:
7         cmd = metrics.ConfusionMatrixDisplay(matrix, display_labels=["Positive","Negative"])
8         ax.text(-0.8,2.1, "Accuracy = "+str(round(accuracy,4))+"\n"+legend+"\nFeatures = "+str(bow.shape[1])+
9             " Elements = "+str(bow.nnz),fontsize=16)
10
11     else:
12         cmd = metrics.ConfusionMatrixDisplay(matrix, display_labels=["1","2","3","4","5"])
13         ax.text(-0.5,6, "Accuracy = "+str(round(accuracy,4))+"\n"+legend+"\nFeatures = "+str(bow.shape[1])+
14             " Elements = "+str(bow.nnz),fontsize=16)
15
16     plt.rc('font', size=18);plt.rc('axes', titlesize=20);plt.rc('axes', labels=14);
17     plt.rc('xtick', labels=12);plt.rc('ytick', labels=12)
18     cmd.plot(ax=ax,cmap="magma_r",values_format='g')
19     ax.set_title(str(name),fontsize=22)
20     plt.savefig("plot/"+data_name+" "+ name+"_matrix.png", bbox_inches="tight")

```

Prediction with a shared dataset (step 2)

Data

```

Entrée [24]: 1 data0=pickle.load(open("french_to_english_film.dataset", "rb"))
2 data1=pickle.load(open("english_to_french_film.dataset", "rb"))

```

```

Entrée [26]: 1 #Define sample number, fold number for the cross validation and features and target columns
2 n = len(data0['rating'].value_counts())
3 cv = 5
4 CV = StratifiedKFold(n_splits=cv, shuffle=True, random_state=1)
5 Xntrain=data0['translation']
6 Yntrain=data0['rating']
7 Xtrain=data1['review']
8 Ytrain=data1['rating']

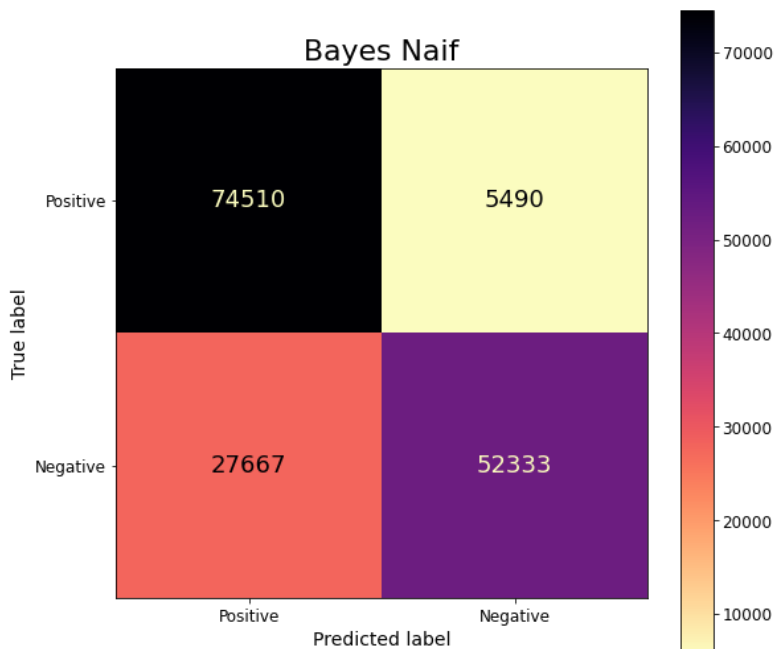
```

Vectorization, training and results

```
Entrée [ ]: 1 #Create vector representing the comment
2 vec = CountVectorizer(lowercase=True, ngram_range=(1, 2), max_df=0.90, min_df=5)
3 bow = vec.fit_transform(Xntrad)
4
5 data_name = "Crosslingual en translated model on en film bow"
6 legend = "No preprocessing. English \nVectorisation with Bow and n-grams=2"
```

```
Entrée [27]: 1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf,bow,Yntrad)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 pipeline = Pipeline([('vec', vec), ('clf', clf)])
12 matrix,accuracy = train(pipeline,Xntrad,Yntrad,Xtrad,Ytrad)
13 plot_matrix(matrix,accuracy,name)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
 Best parameters = {'alpha': 0.3} grid time : 0.3145
 Model Accuracy 1 : 0.79015625
 Model Accuracy 2 : 0.7925625
 Model Accuracy 3 : 0.7963125
 Model Accuracy 4 : 0.7936875
 Model Accuracy 5 : 0.791125
 fit time : 2.6148



Accuracy = 0.7928
 No preprocessing. English
 Vectorisation with Bow and n-grams=2
 Features = 294588 Elements = 19099568

Entrée [28]:

```

1 #Create classifier
2 name = "LogisticRegression"
3 clf = LogisticRegression()
4
5 #Run the gridsearch
6 param_grid = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param=grid(param_grid,clf,bow,Yntrad)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param)
11 pipeline = Pipeline([('vec', vec), ('clf', clf)])
12 matrix,accuracy = train(pipeline,Xntrad,Yntrad,Xtrad,Ytrad)
13 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters = {'C': 0.1} grid time : 5.6857

Model Accuracy 1 : 0.85378125

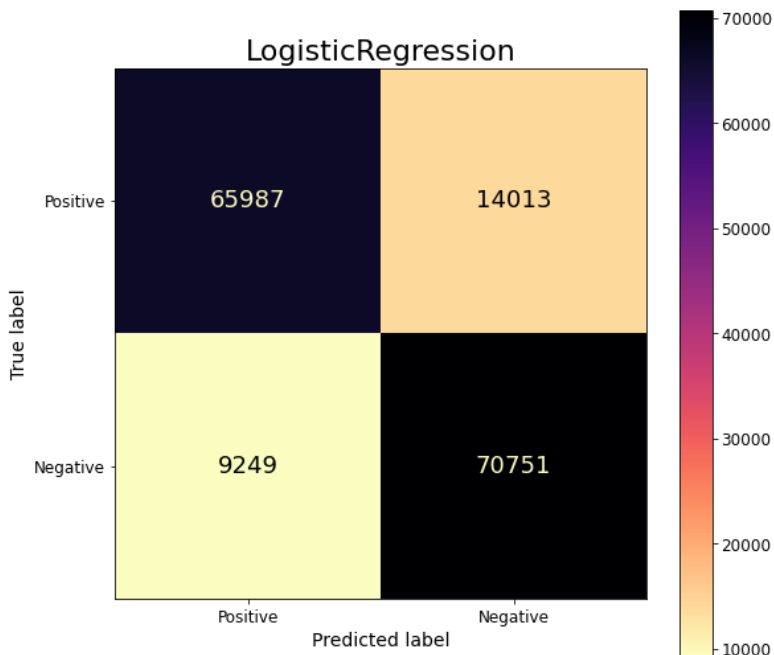
Model Accuracy 2 : 0.85334375

Model Accuracy 3 : 0.8549375

Model Accuracy 4 : 0.85790625

Model Accuracy 5 : 0.85309375

fit time : 3.4446



Accuracy = 0.8546

No preprocessing. English

Vectorisation with Bow and n-grams=2

Features = 294588 Elements = 19099568

Entrée []:

```

1 #Create vector representing the comment
2 vec = TfidfVectorizer(lowercase=True, ngram_range=(1, 2), max_df=0.90, min_df=5)
3 tfidf = vec.fit_transform(Xntrad)
4
5 data_name = "Crosslingual en translated model on en film tfidf"
6 legend = "No preprocessing. English \nVectorisation with Tfidf and n-grams=2"

```

Entrée [29]:

```

1 #Create classifier
2 name = "Bayes Naif"
3 clf = MultinomialNB()
4
5 #Run the gridsearch
6 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
7 param=grid(param_grid,clf,tfidf,Yntrad)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = MultinomialNB(**param)
11 pipeline = Pipeline([('vec', vec), ('clf', clf)])
12 matrix,accuracy = train(pipeline,Xntrad,Yntrad,Xtrad,Ytrad)
13 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best parameters = {'alpha': 0.3} grid time : 0.2676

Model Accuracy 1 : 0.806

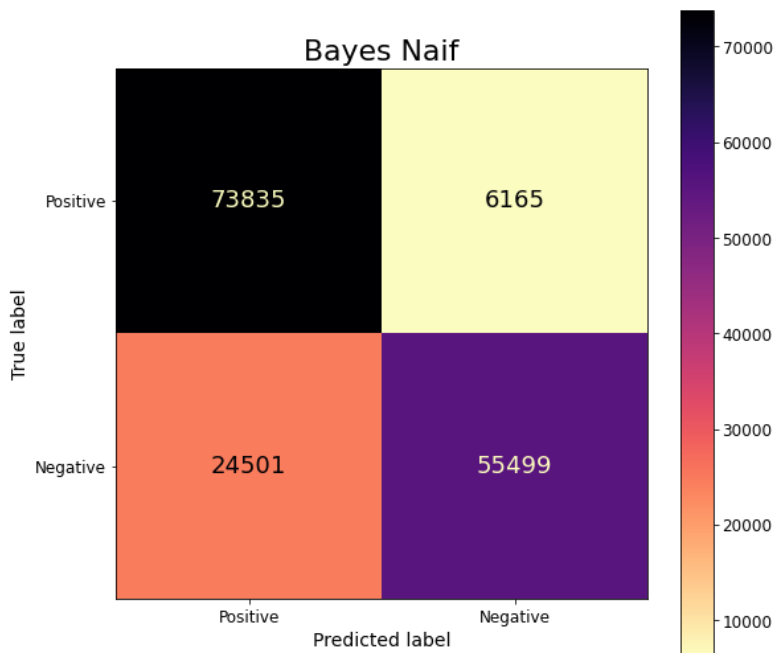
Model Accuracy 2 : 0.806875

Model Accuracy 3 : 0.81284375

Model Accuracy 4 : 0.808625

Model Accuracy 5 : 0.80734375

fit time : 2.6569



Accuracy = 0.8083

No preprocessing. English

Vectorisation with Tfidf and n-grams=2

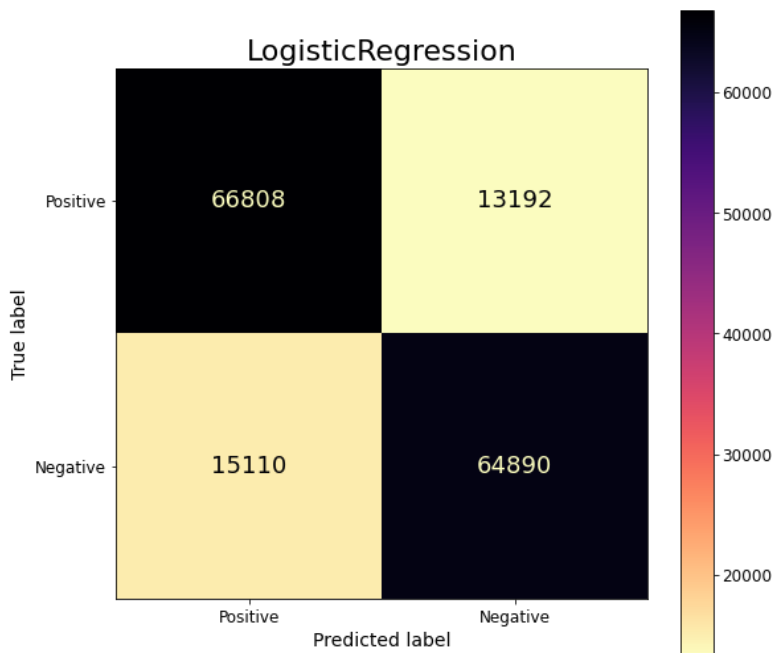
Features = 294588 Elements = 19099568

```

Entrée [30]: 1 #Create classifier
2 name = "LogisticRegression"
3 clf = LogisticRegression()
4
5 #Run the gridsearch
6 param_grid = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
7 param=grid(param_grid,clf,tfidf,Yntrad)
8
9 #Train de classifier with the best parameters then show accuracy and plot confusion matrix
10 clf = LogisticRegression(**param)
11 pipeline = Pipeline([('vec', vec), ('clf', clf)])
12 matrix,accuracy = train(pipeline,Xntrad,Yntrad,Xtrad,Ytrad)
13 plot_matrix(matrix,accuracy,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits
 Best parameters = {'C': 0.1} grid time : 5.2917
 Model Accuracy 1 : 0.82321875
 Model Accuracy 2 : 0.82234375
 Model Accuracy 3 : 0.8264375
 Model Accuracy 4 : 0.8238125
 Model Accuracy 5 : 0.81975
 fit time : 3.2403



Accuracy = 0.8231
 No preprocessing. English
 Vectorisation with Tfidf and n-grams=2
 Features = 294588 Elements = 19099568

Prediction with a more neutral dataset but not fit for the type of data (movies or products) (step 3)

Data

```

Entrée [160]: 1 data1=pickle.load(open("english_to_french_film.dataset", "rb"))
               2 data2=pickle.load(open("english_to_french_product.dataset", "rb"))
               3 data3=pickle.load(open("french_to_english_product.dataset", "rb"))

Entrée [161]: 1 #Set on True if the first dataset used for training (data1) is composed of movie rating (False if it is product rating)
               2 movie = True
               3 if movie:
               4     data2['rating'] = data2['rating'].str.replace("2", "1")
               5     data2['rating'] = data2['rating'].str.replace("4", "5")
               6     data2 = data2[data2.rating != "3"]
               7     data2=data2.reset_index(drop=True)
               8     data2['rating'] = data2['rating'].str.replace("1", "0")
               9     data2['rating'] = data2['rating'].str.replace("5", "1")
              10
              11     data3['rating'] = data3['rating'].str.replace("2", "1")
              12     data3['rating'] = data3['rating'].str.replace("4", "5")
              13     data3 = data3[data3.rating != "3"]
              14     data3=data3.reset_index(drop=True)
              15     data3['rating'] = data3['rating'].str.replace("1", "0")
              16     data3['rating'] = data3['rating'].str.replace("5", "1")
              17 else:
              18     data1['rating'] = data1['rating'].str.replace("2", "1")
              19     data1['rating'] = data1['rating'].str.replace("4", "5")
              20     data1 = data1[data1.rating != "3"]
              21     data1=data1.reset_index(drop=True)
              22     data1['rating'] = data1['rating'].str.replace("1", "0")
              23     data1['rating'] = data1['rating'].str.replace("5", "1")

Entrée [163]: 1 #Define sample number, fold number for the cross validation and features and target columns
               2 n = len(data1['rating'].value_counts())
               3 cv = 5
               4 CV = StratifiedKFold(n_splits=cv, shuffle=True, random_state=1)
               5 Xntrad=data1['translation']
               6 Yntrad=data1['rating']
               7 X2=data2['translation']
               8 Y2=data2['rating']
               9 X3=data3['review']
              10 Y3=data3['rating']

```

Vectorization, training and results

Entrée [166]:

```

1 #Create vector representing the comment
2 vec = CountVectorizer(lowercase=True, ngram_range=(1, 2), max_df=0.90, min_df=5)
3 bow = vec.fit_transform(Xntrad)
4
5 data_name = "Crosslingual neutral fr_product_trad on translated film model bow"
6 legend = "No preprocessing. French Translated \nVectorisation with Bow and n-grams=2"
7
8 #Create classifier
9 name = "Bayes Naif"
10 clf = MultinomialNB()
11
12 #Run the gridsearch
13 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
14 param=grid(param_grid,clf,bow,Yntrad)
15
16 #Train de classifier with the best parameters then show accuracies and plot confusion matrices
17 clf = MultinomialNB(**param)
18 pipeline = Pipeline([('vec', vec), ('clf', clf)])
19 matrix1,accuracy1,matrix2,accuracy2 = train_cross(pipeline,Xntrad,Yntrad,X2,Y2,X3,Y3)
20 plot_matrix(matrix1,accuracy1,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best parameters = {'alpha': 0.7} grid time : 0.4391

Model Accuracy 1 : 0.74203125

Model Accuracy 1 : 0.74959375

Model Accuracy 2 : 0.74015625

Model Accuracy 2 : 0.74765625

Model Accuracy 3 : 0.74353125

Model Accuracy 3 : 0.7505625

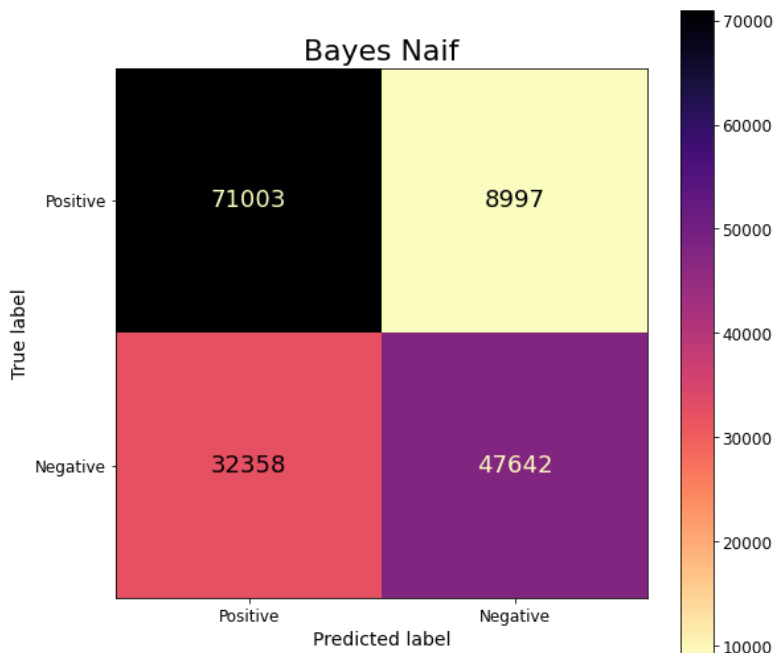
Model Accuracy 4 : 0.73771875

Model Accuracy 4 : 0.75171875

Model Accuracy 5 : 0.74421875

Model Accuracy 5 : 0.7503125

fit time : 3.9059



Accuracy = 0.7415

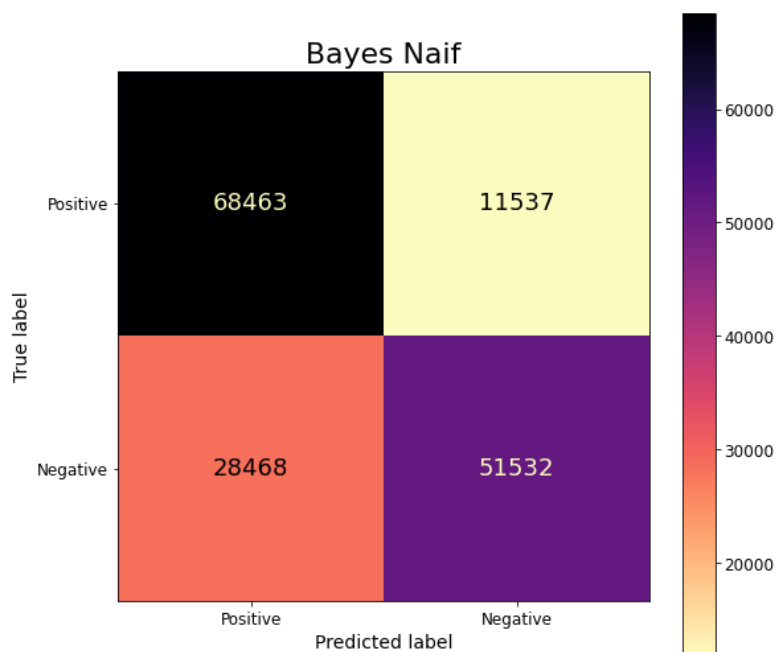
No preprocessing. French Translated

Vectorisation with Bow and n-grams=2

Features = 427429 Elements = 35124053

Entrée [167]:

```
1 #Change names
2 data_name = "Crosslingual neutral fr_product_review on translated film model bow"
3 legend = "No preprocessing. French \nVectorisation with Bow and n-grams=2"
4
5 #Plot the second matrix
6 plot_matrix(matrix2,accuracy2,name)
```



Accuracy = 0.75
No preprocessing. French
Vectorisation with Bow and n-grams=2
Features = 427429 Elements = 35124053

Entrée [168]:

```

1 #Create vector representing the comment
2 vec = CountVectorizer(lowercase=True, ngram_range=(1, 2), max_df=0.90, min_df=5)
3 bow = vec.fit_transform(Xntraded)
4
5 data_name = "Crosslingual neutral fr_product_trad on translated film model bow"
6 legend = "No preprocessing. French Translated \nVectorisation with Bow and n-grams=2"
7
8 #Create classifier
9 name = "LogisticRegression"
10 clf = LogisticRegression()
11
12 #Run the gridsearch
13 param_grid = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
14 param=grid_search(param_grid,clf,bow,Yntraded)
15
16 #Train de classifier with the best parameters then show accuracies and plot confusion matrices
17 clf = LogisticRegression(**param)
18 pipeline = Pipeline([('vec', vec), ('clf', clf)])
19 matrix1,accuracy1,matrix2,accuracy2 = train_cross(pipeline,Xntraded,Yntraded,X2,Y2,X3,Y3)
20 plot_matrix(matrix1,accuracy1,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters = {'C': 0.05} grid time : 11.0274

Model Accuracy 1 : 0.76871875

Model Accuracy 1 : 0.76046875

Model Accuracy 2 : 0.759625

Model Accuracy 2 : 0.75725

Model Accuracy 3 : 0.7651875

Model Accuracy 3 : 0.7560625

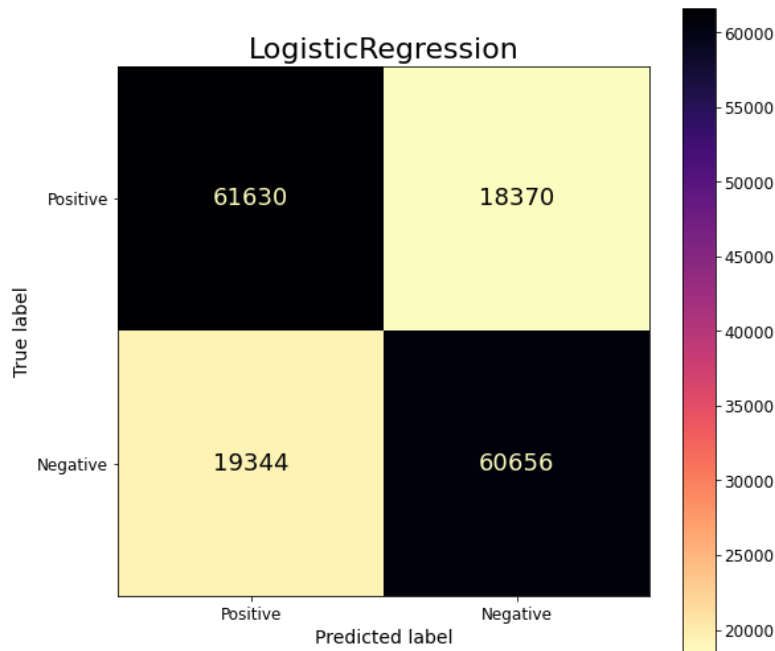
Model Accuracy 4 : 0.76025

Model Accuracy 4 : 0.7606875

Model Accuracy 5 : 0.76765625

Model Accuracy 5 : 0.7578125

fit time : 5.5011



Accuracy = 0.7643

No preprocessing. French Translated

Vectorisation with Bow and n-grams=2

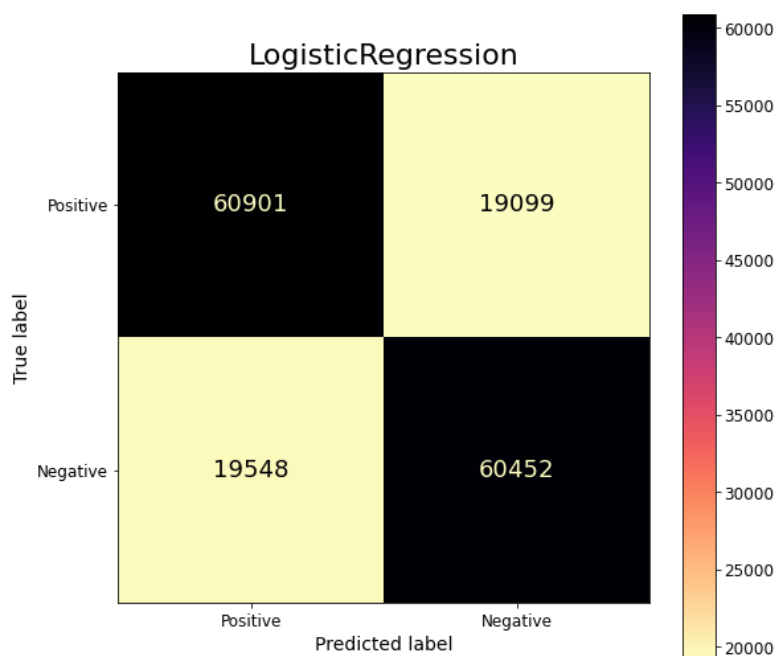
Features = 427429 Elements = 35124053

Entrée [169]:

```

1 #Change names
2 data_name = "Crosslingual neutral fr_product_review on translated film model bow"
3 legend = "No preprocessing. French \nVectorisation with Bow and n-grams=2"
4
5 #plot second matrix
6 plot_matrix(matrix2,accuracy2,name)

```



Accuracy = 0.7585
 No preprocessing. French
 Vectorisation with Bow and n-grams=2
 Features = 427429 Elements = 35124053

Entrée [170]:

```

1 #Create vector representing the comment
2 vec = TfidfVectorizer(lowercase=True, ngram_range=(1, 2), max_df=0.90, min_df=5)
3 tfidf = vec.fit_transform(Xntrad)
4
5 data_name = "Crosslingual neutral fr_product_trad on translated film model tfidf"
6 legend = "No preprocessing. French Translated \nVectorisation with Tfidf and n-grams=2"
7
8 #Create classifier
9 name = "Bayes Naif"
10 clf = MultinomialNB()
11
12 #Run the gridsearch
13 param_grid = {'alpha':[0.1,0.3,0.7,1,2,3,4,5,7,10]}
14 param=grid(param_grid,clf,tfidf,Yntrad)
15
16 #Train de classifier with the best parameters then show accuracies and plot confusion matrices
17 clf = MultinomialNB(**param)
18 pipeline = Pipeline([('vec', vec), ('clf', clf)])
19 matrix1,accuracy1,matrix2,accuracy2 = train_cross(pipeline,Xntrad,Yntrad,X2,Y2,X3,Y3)
20 plot_matrix(matrix1,accuracy1,name)

```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best parameters = {'alpha': 0.7} grid time : 0.408

Model Accuracy 1 : 0.74234375

Model Accuracy 1 : 0.74634375

Model Accuracy 2 : 0.7405

Model Accuracy 2 : 0.74659375

Model Accuracy 3 : 0.74434375

Model Accuracy 3 : 0.747125

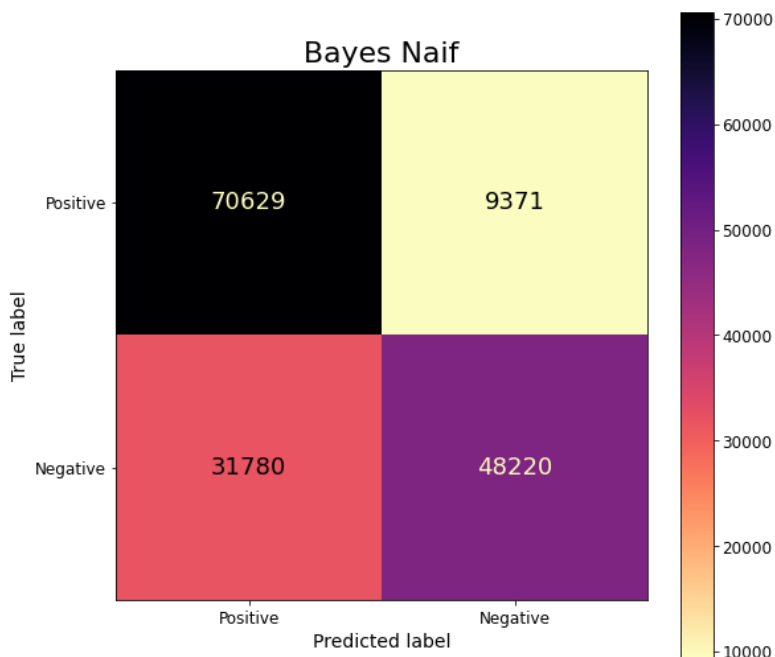
Model Accuracy 4 : 0.74140625

Model Accuracy 4 : 0.746625

Model Accuracy 5 : 0.7454375

Model Accuracy 5 : 0.7458125

fit time : 3.6011



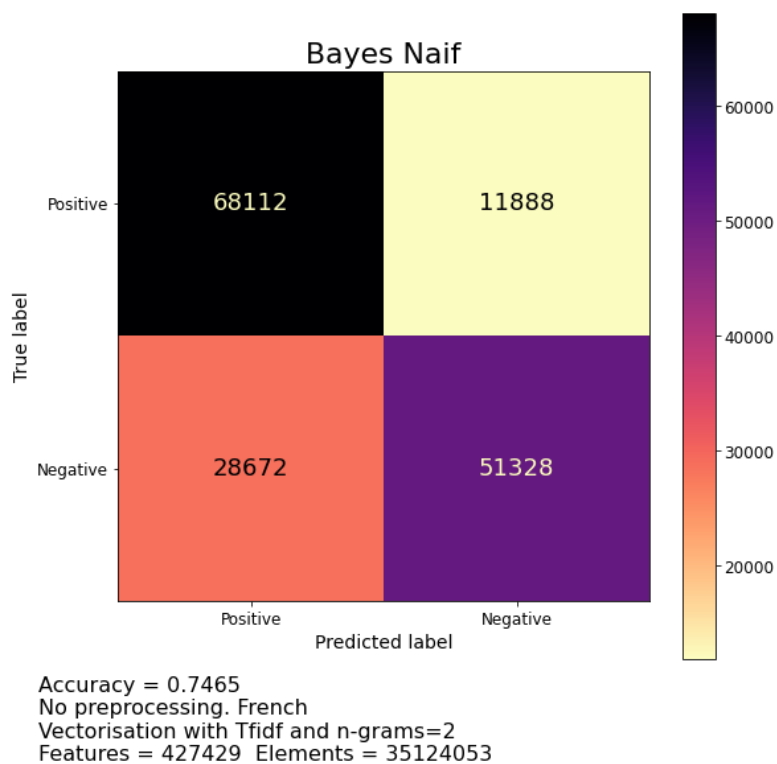
Accuracy = 0.7428

No preprocessing. French Translated

Vectorisation with Tfidf and n-grams=2

Features = 427429 Elements = 35124053

```
Entrée [171]: 1 #Change names
2 data_name = "Crosslingual neutral fr_product_review on translated film model tfidf"
3 legend = "No preprocessing. French \nVectorisation with Tfidf and n-grams=2"
4
5 #Show second matrix
6 plot_matrix(matrix2,accuracy2,name)
```



Entrée [172]:

```

1 #Create vector representing the comment
2 vec = CountVectorizer(lowercase=True, ngram_range=(1, 2), max_df=0.90, min_df=5)
3 tfidf = vec.fit_transform(Xntrad)
4
5 data_name = "Crosslingual neutral fr_product_trad on translated film model tfidf"
6 legend = "No preprocessing. French Translated \nVectorisation with Tfidf and n-grams=2"
7
8 #Create classifier
9 name = "LogisticRegression"
10 clf = LogisticRegression()
11
12 #Run the gridsearch
13 param_grid = {'C':[0.01,0.05,0.1,0.5,1,5,10,50]}
14 param=grid(param_grid,clf,tfidf,Yntrad)
15
16 #Train de classifier with the best parameters then show accuracies and plot confusion matrices
17 clf = LogisticRegression(**param)
18 pipeline = Pipeline([('vec', vec), ('clf', clf)])
19 matrix1,accuracy1,matrix2,accuracy2 = train_cross(pipeline,Xntrad,Yntrad,X2,Y2,X3,Y3)
20 plot_matrix(matrix1,accuracy1,name)

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters = {'C': 10} grid time : 5.2272

Model Accuracy 1 : 0.7721875

Model Accuracy 1 : 0.76909375

Model Accuracy 2 : 0.76590625

Model Accuracy 2 : 0.764125

Model Accuracy 3 : 0.76496875

Model Accuracy 3 : 0.75878125

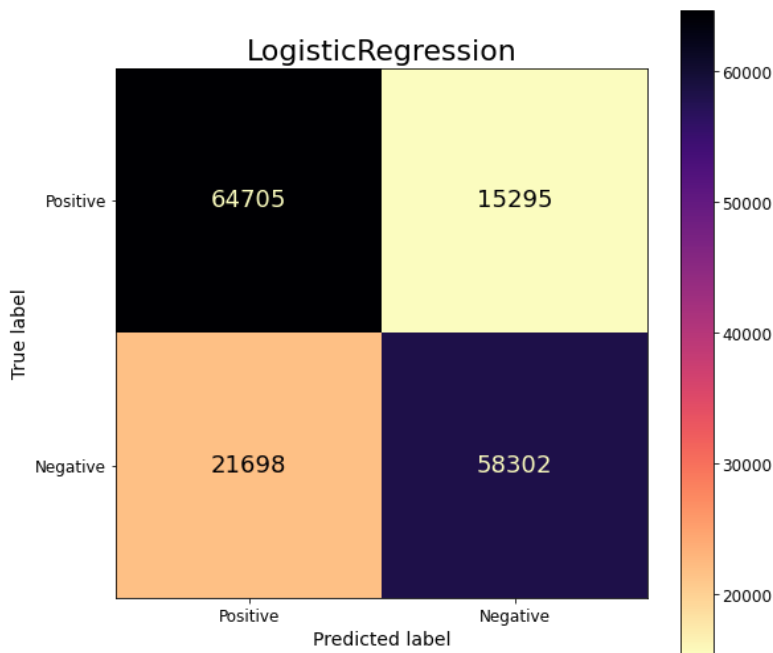
Model Accuracy 4 : 0.76946875

Model Accuracy 4 : 0.76871875

Model Accuracy 5 : 0.7714375

Model Accuracy 5 : 0.763625

fit time : 4.8637



Accuracy = 0.7688

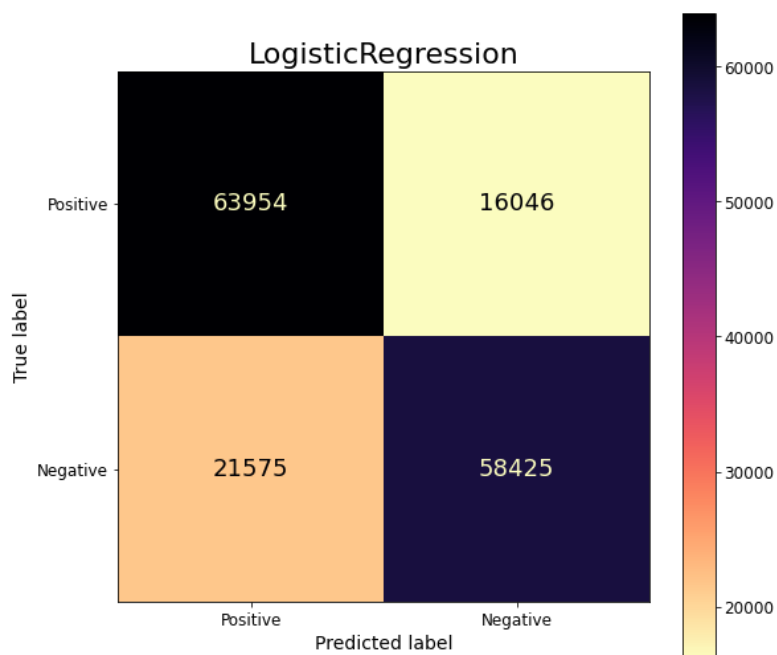
No preprocessing. French Translated

Vectorisation with Tfidf and n-grams=2

Features = 427429 Elements = 35124053

Entrée [173]:

```
1 #Change names
2 data_name = "Crosslingual neutral fr_product_review on translated film model tfidf"
3 legend = "No preprocessing. French \nVectorisation with Tfidf and n-grams=2"
4
5 #Show second matrix
6 plot_matrix(matrix2,accuracy2,name)
```



Accuracy = 0.7649
No preprocessing. French
Vectorisation with Tfidf and n-grams=2
Features = 427429 Elements = 35124053