Introduction to Machine Learning for Natural Language Processing: Analysis

Daniel Russo

June 2021

1 Introduction

Is it possible to easily fool a Naive Bayes classifier (or any other classification algorithm) that perform the task of Authorship Attribution using encrypting or automatic translation methods? According to the state-of-the art, this task is known as *Author Obfuscation Task* and aims to automatically create a paraphrase of a given text so that eventually a classifier will not be able to detect its author. All the various approaches that have been proposed throughout the years pursue the best obfuscation trying to conciliate three main aspects: (1) the capacity to fool a classifier, (2) to keep intact the semantics of the original text, and, finally, (3) to go unnoticed by human eyes.

In the next sections two experiments that try to address this challenge will be proposed: on the one hand, the first experiment is based on a *"brute force"* encryption method that tries to fool the classifier by obscuring the main features of the text; on the other hand, the second experiment attempts to create an acceptable paraphrase according to the three Obfuscation criteria mentioned above.

2 Data and model

This project is based on Practical one of the Master's course: no changes were made to the Naive Bayes classifier already implemented ($\alpha = 0.0001$, $\mu = 0$, $\sigma = 0.2$). Data, on the other hand, has been slightly modified: in order to avoid biases, `shortemma.txt` was removed and the NB classifier was trained on the remaining books of the four authors: *Emma* (Austen), *Alice's Adventures in Wonderland* (Carroll), *The Jungle Book* (Kipling), *The Wind in the Willows* (Grahame). As test file, both for the Authorship Attribution and the Obfuscation task, *Persuasion* by Jane Austen was employed.

3 Experimental setup

As mentioned in the introductory section of this report, two experiments were carried out. Technical details and results of each experiment will be discussed below.

3.1 Experiment 1: Encryption method

As the classifier used in this project is a Naive Bayes one, the fact that the features used as input of the classifier are independent can be taken for granted. Thus, the main idea of the method used in this experiment is to *"obfuscate"* the primary features related to the author of the test file (calculated during the training phase) in order to diminish the probability to detect the right author by adding new and unknown words which will not be present in the vocabulary.

For this purpose, *Caesar cipher* encryption technique has been adopted to substitute a set of features with the corresponding encryption. According to this simple algorithm, the new *"word"* is obtained by the substitution of each character x with another one z calculated by adding to the position of x in the alphabet (in this case the Unicode list) a fixed number, called *shift*. The value of the *shift* used in this experiment is 4.

The sets of features are extracted from the ordered list (from the most to the least probable) of conditional probabilities calculated during the training of the classifier. Different test were conducted: The Naive Bayes algorithm, trained on various features type (words, n-gram of characters ($length = \{3, 4, 5, 7, 10\}$)) has been tested on gradually increasing number of encrypted features in the test file ($number\ of\ encrypted\ features = \{0, 10, 100, 250, 500\}$).

Two necessary clarifications must be done: (1) I am well aware of the fact that the *Caesar cipher* encryption technique is weak and easily hackable using a brute-force attack, however every encryption technique, from the simplest to the most complex, would have been suitable for the project purposes; (2) The method proposed in this section, whatever results leads to, is able to potentially satisfy only the first characteristic that the *"perfect"* obfuscated text must have - the capacity tho fool the classifier - as described in the Introduction section.

3.2 Experiment 2: Obfuscation through translation method

The second experiment tackle the Author Obfuscation task trying to create a paraphrase of a given text that maintain as much as possible the original meaning, and whose rewrite seems plausible to a human eye. In order to achieve that, the obfuscated text is automatically created thanks to a multi-translation process of the test file.

The idea behind this experiment is very simple: given an input text t and a set of foreign languages $\theta(l)$, whose cordiality is equal or greater than 1, the English text is used as input of a pipeline translation process in which the output of a translation between two languages ($\theta_i(l) \rightarrow \theta_j(l)$) is used as input for a translation in the next language of the set ($\theta_1(l) \rightarrow \theta_2(l) \rightarrow ... \rightarrow \theta_n(l)$). Once this process ends the ultimate text version (the output of the pipeline) is translated back into English.

Four languages have been used for this experiment according to four main criteria: belonging to the same linguistic group, in this case the Germanic languages group (**Dutch**); high lexical similarity coefficient (**French**, which also belongs to a different language group, the Romance Languages); an alphabet different from the Latin one (**Russian**); an analytic language like English (**Chinese**, which does not use an alphabet but logograms instead).

The test file has been obfuscated in six different ways following the pipeline translation process described above. Those processes are:

$$English \rightarrow Dutch \rightarrow English$$

$$English \rightarrow French \rightarrow English$$

$$English \rightarrow Russian \rightarrow English$$

$$English \rightarrow Chinese \rightarrow English$$

$$English \rightarrow Dutch \rightarrow French \rightarrow English$$

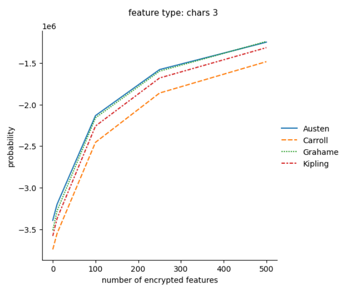
$$English \rightarrow Russian \rightarrow Chinese \rightarrow English$$

In order to speed up the numerous translations pipelines, a multiprocessing approach has been adopted instantiating as many concurrent processes as the number of CPU of the machine used (precisely, the cardinality of the *pool* of processes used for this experiment was 8).

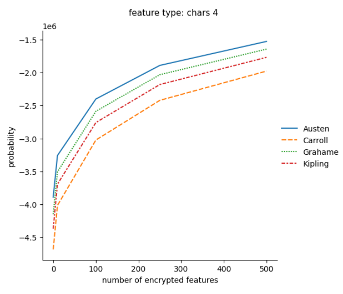
Finally, as in the previous experiment, each obfuscated file has been tested on different feature types, words and n-gram of characters ($length$ = {3, 4, 5, 7, 10}).

4 Results

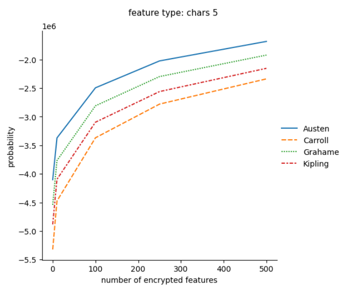
The first experiments manage to fool the Naive Bayes classifier only in one of the six tests that were carried out: as shown in Figure 1a, training on n-grams of characters with length equal to 3 as features and encrypting the first 500 most probable, the algorithm detect Grahame and not Austen as author of *Persuasion*. Even though all the other test fail the challenge, from the graphs in Figure 1 can be seen that, as expected, the number of encrypted features and the value of the probabilities are inversely proportional: while increasing the encrypted features all the probability scores become more and more smaller. Finally, results show that the increase of n-grams length lead the classifier to overfitting, becoming overconfident about its final decision.



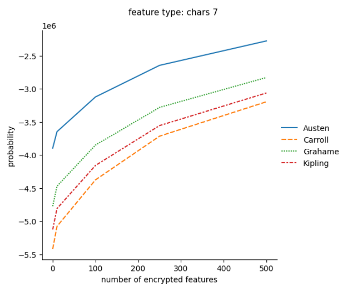
(a)



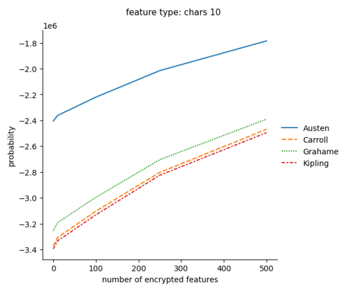
(b)



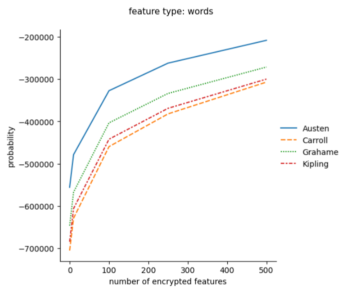
(c)



(d)



(e)



(f)

Figure 1: Results of Experiment 1

None of the tests held in the second experiment was able to fool the Naive Bayes classifier, thus *Austen* was always detected as the author of each obfuscated file no matter what feature type was involved in the training phase. As in the experiment before, the obfuscation processes allowed to reduce the value of probabilities. Moreover, results (Figure 2) shows a similar behaviour of the various translations independently from the feature type used: obfuscating the text by translating the original text in *Dutch* (same linguistic group), *French* (high lexical similarity), and in a combination of the two (*Dutch* $\rightarrow$ *French*) has always been less effective in terms of final probability scores than using languages with a different alphabet, like *Russian*, or that doesn't have an alphabet at all, *Japanese*.

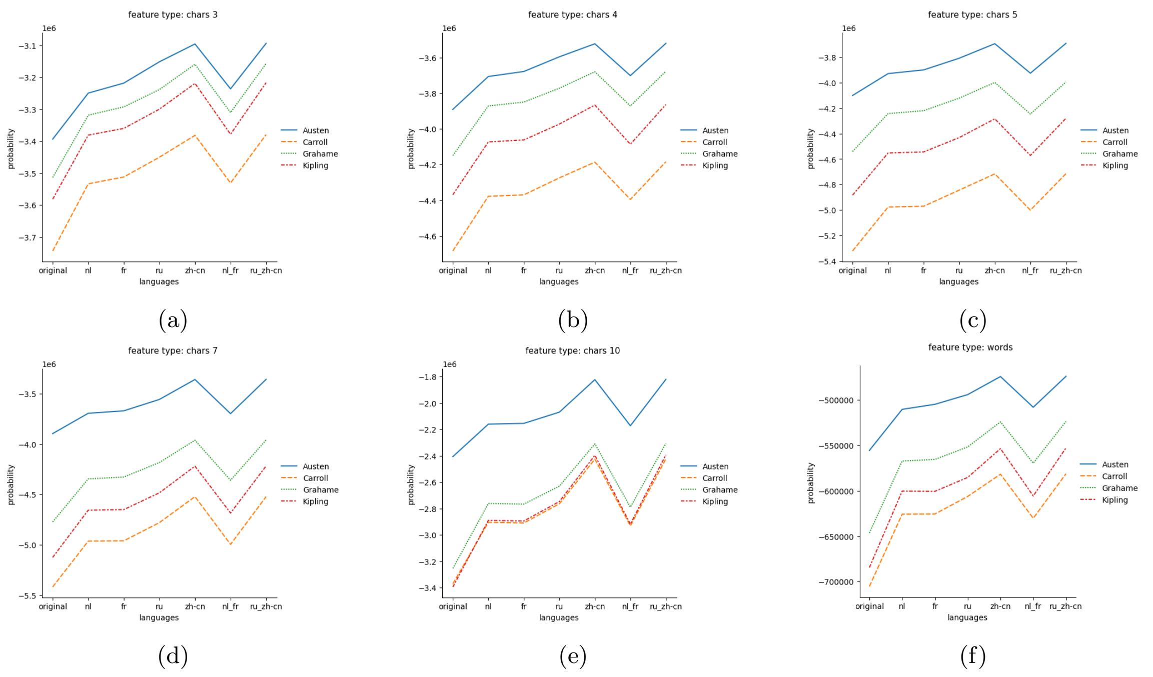


Figure 2: Results of Experiment 2

5 Evaluation

According to the state of the art, the quality of the obfuscation process can not only be assessed automatically: part of the evaluation must be held by humans. In order to evaluate whether an obfuscation approach is acceptable or not, this must be assessed according to three dimensions (Hagen, Potthast, and Stein 2017) which are: **Safety**, that evaluate whether the original author can not be inferred from the obfuscated text; **Soundness**, which appraises if the original text *lexically entails* the obfuscated one; **Sensibleness**, to estimate how the obfuscated text is well written and properly camouflaged to human eyes. The safety of the obfuscation approaches described in this report has been assessed simply by checking whether the Naive Bayes classifiers fails in the right author detection.

As pointed out in the results section, the only obfuscation approach that can be considered safe is the *Encryption method* with more than 500 encrypted features, processed as n-grams of three characters.

Respecting to Sensibleness and Soundness, evaluative results vary between the two experiments. The encryption method proposed in the first one fails both in creating an obfuscated text that maintains the same meaning of the original one, and in hiding to human eyes the fact that it has been obfuscated: in fact, the substitution of sets of features in the text with their meaningless encryption lead to a text no longer fully readable and understandable. On the other hand, the translation method proposed in the second experiment was evaluated by humans: taking inspiration from Hagen, Potthast, and Stein 2017 a Google form was used to evaluate Sensibleness and Soundness of the obfuscated texts. From the original text were randomly extracted three chunks of text and then translated according to the pipeline translation processes listed in the Experimental setup section. Every human assessor was asked to read the resulting obfuscated chunks of text and, for each one give a school grade on a scale from 1 (excellent) to 5 (fail) after highlighting all the possible errors in them (**Soundness assessment**). Eventually, the original version of the text was provided and evaluators had to compare it with each of the obfuscated chunks and to say whether the paraphrase was "correct", "passable" or "incorrect" (**Sensibleness assessment**). Results of the evaluation of five human assessors are summarized in Table 1: Soundness values were obtained by averaging the mean of all the school grades assigned; Sensibleness was assessed by considering the most frequent of all the judgments assigned. The best scores in Soundness were obtained by obfuscations that involved Dutch and French in the translation processes ($Soundness < 2,5$). This applies also to the Sensibleness evaluation, which shows higher quality of translation in correspondence of the just mentioned languages.

Translations	**Soundness**	**Sensibleness**
Dutch	2	passable
French	2,27	correct
Russian	2,8	incorrect
Chinese	3,7	incorrect
Dutch $\rightarrow$ French	2,1	correct
Russian $\rightarrow$ Chinese	3,7	incorrect

Table 1: Evaluation results of Experiment 2

6 Discussion

As explained in the sections before, both experiments led to unsafe obfuscated text in the majority of the tests that were carried out. One of the problem of the first experiment is related to the training data used for this project in which there was only one book per author. Thus, the main features selected in order to be obfuscated in the test file were not strictly related to the author we were dealing with, *Jane Austen* in our case, but to single book of Emma. Possible solutions that can help to improve the results and, hopefully, fool the classifier are: ameliorate the training set by adding various book per author, so that the system is able to find the most relevant author features; change the selection process of the most relevant features, replacing the *Bag-of-words* model, which relies on the terms frequency, with a selection base on the *term frequency–inverse document frequency (tf-idf)* value.

As said before, in the second experiment all the tests held didn't success in fooling the classifier. Two of the reasons that caused the failure of the obfuscation task could be related to: (1) shared characteristic between the original language (English) and the languages used in the translation processes; (2) characteristic of the computational model that automatically translates one language into another. English shares with French and Dutch various common features (words, lexical cognates etc.) due to historical events crucial for its development. The probability that these shared features between languages remain unvaried at the end of the obfuscation through translation process is very high, thus the Naive Bayes classifier will benefits from it during the testing process recognizing and linking them to the right author. Moreover, even though the computational model used for the automatic translation - in this project the *Google's Neural Machine Translation System* (Wu et al. 2016), a deep LSTM network with an attention module - is able to return good translation taking into consideration not only the single words of the sentences but also the syntax of it, when translating again the text into English in the majority of the cases tends to use the same words of the original text. This behaviour applies not only for French and Dutch, similar to English as described before, but also for Russian and Chinese, and again assists the Naive Bayes classifier in recognizing known words learnt during its training phase.

7 Acknowledgements

I would like to thank Beatrice, Claudia, Giovana, Giulia, and Maria Luisa for their help in evaluating the Task.

8 Appendix

All the input data, the output files, as well as the whole code and results, can be found in this GitHub repository.

```python
import re
from deep_translator import GoogleTranslator
from multiprocessing.dummy import Pool as ThreadPool

class Obfuscator:

    def __init__(self,document,author,probabilities,features):
        self.document = document
        self.author = author
        self.probabilities = probabilities
        self.features = features

    def set_document(self,document):
        self.document = document

    def get_document(self):
        return self.document

    def get_main_features(self,author,n):
        cps = {}

        for term, probs in self.probabilities.items():
            cps[term] = probs[author]

        cps = sorted(cps, key=cps.get, reverse=True)
        return list(cps[:n])

    def __encrypt(self,text,shift):
        crypted_text = ""
        for idx in range(len(text)):
            char = text[idx]
            crypted_text += chr(ord(char) + shift)

        return crypted_text

    def __decrypt(self,text,shift):
        original_text = ""
        for idx in range(len(text)):
            char = text[idx]
            original_text += chr(ord(char) - shift)

        return original_text

    def preprocess_doc(self,doc):
        file = open(doc, "r", encoding="utf-8")
        text = file.read()
        file.close()
        chunks = text.split(".")
        return chunks

    def translate(self,languages,text):
        try:
            translated = ""
            for lang in languages:
                translated = GoogleTranslator(target=lang).translate(text)
            translated = GoogleTranslator(target='en').translate(translated)
            return translated
        except:
            return text

    def obfuscate(self, type, n_features=None, file=None, languages=None):

        print("\n### OBFUSCATION PROCESS IN PROGRESS ###\n")

        if type == "encrypt":
            print("\n>>> EXPERIMENT N.1\n")
            features = self.get_main_features("Austen",n_features)
            print("Number of features to encrypt: ",len(features))
            print(">>> processing...\n")
            with open(self.document, "r", encoding="utf-8") as file:
                obf_doc = file.read()
                file.close()

            for feature in features:
                obf_f = self.__encrypt(feature,4)
                if self.features == "words":
                    obf_doc = re.sub(r"\b" + feature + r"\b",obf_f,obf_doc)
                else:
                    obf_doc = re.sub(feature, obf_f, obf_doc)

            with open("encrypted_doc.txt", "w", encoding="utf-8") as file:
                file.write(obf_doc)
                file.close()

        elif type == "translate":
            print("\n>>> EXPERIMENT N.2\n")
            print(">>> Translation pipeline: en -> " + " -> ".join(languages) + " -> en")
            print(">>> Document: " + file)
            print(">>> processing...\n")
            pool = ThreadPool()
            document = self.preprocess_doc(file)
            lang = [languages]*len(document)
            result = pool.starmap(self.translate, zip(lang,document))
            pool.close()
            pool.join()
            ob_text = ".".join(result)
            path = "translated/obf_file(" + "_".join(languages) + ").txt"
            ob_file = open(path, "w", encoding="utf-8")
            ob_file.write(ob_text)
            ob_file.close()
            print(">>> Output file: " + path)

        print("\n### OBFUSCATION PROCESS ENDED ###\n")

    def success(self,probabilities):
        return probabilities[0][0] != self.author
```
Listing 1: Obfuscator Class.

```python
"""Authorship Attribution

Usage:
  attribution.py --words <filename>
  attribution.py --chars=<n> <filename>
  attribution.py (-h | --help)
  attribution.py --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --words
  --chars=<kn>  Length of char ngram [default: 3].

"""

import sys
import os
import math
from utils import process_document_words, process_document_ngrams, get_documents, extract_vocab,
    top_cond_probs_by_author
from docopt import docopt
import numpy as np
from obfuscator import Obfuscator
import seaborn as sns
import pandas as pd
from os import listdir
from os.path import isfile, join

if __name__ == '__main__':
    arguments = docopt(__doc__, version='Authorship Attribution 1.1')


'''Default values for hyperparameters'''
feature_type = "words"
ngram_size = 3
testfile = "data/test/persuasion.txt"

if arguments["--words"]:
    feature_type = "words"
elif arguments["--chars"]:
    feature_type = "chars"
    ngram_size = int(arguments["--chars"])
testfile = arguments["<filename>"]


alpha = 0.0001
mu, sigma = 0, 0.2 # mean and standard deviation
classes = ["Austen", "Carroll", "Grahame", "Kipling"]
documents = get_documents(feature_type, ngram_size)

def count_docs(documents):
    return len(documents)

def count_docs_in_class(documents, c):
    count=0
    for values in documents.values():
        if values[0] == c:
            count+=1
    return count

def concatenate_text_of_all_docs_in_class(documents,c):
    words_in_class = {}
    for d,values in documents.items():
        if values[0] == c:
            words_in_class.update(values[2])
    return words_in_class

def train_naive_bayes(classes, documents):
    vocabulary = extract_vocab(documents)
    conditional_probabilities = {}
    for t in vocabulary:
        conditional_probabilities[t] = {}
    priors = {}
    print("\n\n***\nCalculating priors and conditional probabilities for each class...\n***")
    for c in classes:
         priors[c] = count_docs_in_class(documents,c) / count_docs(documents)
         print("\nPrior for",c,priors[c])
         class_size = count_docs_in_class(documents, c)
         print("In class",c,"we have",class_size,"document(s).")
         words_in_class = concatenate_text_of_all_docs_in_class(documents,c)
         #print(c,words_in_class)
         print("Calculating conditional probabilities for the vocabulary.")
         denominator = sum(words_in_class.values())
         for t in vocabulary:
             if t in words_in_class:
                 conditional_probabilities[t][c] = (words_in_class[t] + alpha) / (denominator * (1
 + alpha))
                 # print(t,c,words_in_class[t],denominator,conditional_probabilities[t][c])
             else:
                conditional_probabilities[t][c] = (0 + alpha) / (denominator * (1 + alpha))
    return vocabulary, priors, conditional_probabilities

def apply_naive_bayes(classes, vocabulary, priors, conditional_probabilities, test_document):
    scores = {}

    if feature_type == "chars":
        author, doc_length, words = process_document_ngrams(test_document,ngram_size)
    elif feature_type == "words":
        author, doc_length, words = process_document_words(test_document)
    for c in classes:
        scores[c] = math.log(priors[c])
        for t in words:
            if t in conditional_probabilities:
                for i in range(words[t]):
                    scores[c] += math.log(conditional_probabilities[t][c])
    print("\n\nNow printing scores in descending order:")
    probabilities = []
    for author in sorted(scores, key=scores.get, reverse=True):
        probabilities.append((author,scores[author]))
        print(author,"score:",scores[author])
    return probabilities

# ===================
# author: Daniel Russo
# ===================

def plot_results(data, exp, xlabel, ylabel):
    df = pd.DataFrame(data)
    sns_plot = sns.relplot(data=df, kind="line")
    if feature_type == "chars":
        sns_plot.fig.suptitle("feature type: " + feature_type + " " + str(ngram_size), fontsize
    =11)
        file_name = feature_type + "_" + str(ngram_size)
    else:
        sns_plot.fig.suptitle("feature type: " + feature_type, fontsize=11)
        file_name = feature_type
    sns_plot.fig.subplots_adjust(top=0.9);
    sns_plot.set(xlabel=xlabel, ylabel=ylabel)
    sns_plot.savefig("results/output_" + exp + "_" + file_name + ".png")

def experiment_1(classes, file, conditional_probabilities, feature_type, vocabulary, priors):
    test_features = [0, 10, 100, 250, 500]
    obf_successes = []
    prob_results = {author: {0: None, 10: None, 100: None, 250: None, 500: None} for author in
    classes}
    obf = Obfuscator(file, "Austen", conditional_probabilities, feature_type)
    obf_file = "encrypted_doc.txt"

    for num in test_features:
        obf.obfuscate(type="encrypt", n_features=num)
        probs = apply_naive_bayes(classes, vocabulary, priors, conditional_probabilities, obf_file
    )
        obf_successes.append(obf.success(probs))
        for p in probs:
            prob_results[p[0]][num] = p[1]

    # plotting results
    plot_results(prob_results, "exp1", "number of encrypted features", "probability")

    return obf_successes


def experiment_2(file, conditional_probabilities, feature_type, classes, vocabulary, priors):
    obf = Obfuscator(file, "Austen", conditional_probabilities, feature_type)
    languages = [["nl"],["fr"],["ru"],["zh-cn"],["nl","fr"],["ru","zh-cn"]]

    # change the index of languages[index] in order to translate in the corresponding language
    # comment the next line to skip the obfuscation(translation) part and to use already existing
    obfuscated files
    obf.obfuscate(type="translate",file=file,languages=languages[0])

    obf_files = [f for f in listdir("translated") if isfile(join("translated", f))]
    prob_results = {author: {"original": None, "nl": None, "fr": None, "ru": None, "zh-cn": None}
    for author in classes}
    obf_successes = []

    # testing on the original file and storing probabilities
    original_probs = apply_naive_bayes(classes, vocabulary, priors, conditional_probabilities,
    file)
    for p in original_probs:
        prob_results[p[0]]["original"] = p[1]

    # testing the obfuscated files and storing probabilities
    for file in obf_files:
        probs = apply_naive_bayes(classes, vocabulary, priors, conditional_probabilities, "
    translated/"+file)
        obf_successes.append(obf.success(probs))
        for p in probs:
            lang = file.split("(")[1].split(")")[0]
            prob_results[p[0]][lang] = p[1]

    # plotting results
    plot_results(prob_results,"exp2","languages","probability")

    return obf_successes


vocabulary, priors, conditional_probabilities = train_naive_bayes(classes, documents)

for author in classes:
    print("\nBest features for",author)
    top_cond_probs_by_author(conditional_probabilities, author, 10)

# N.B. without obfuscation
apply_naive_bayes(classes, vocabulary, priors, conditional_probabilities, testfile)

# running experiments
results_exp1 = experiment_1(classes,testfile,conditional_probabilities,feature_type,vocabulary,
    priors)
results_exp2 = experiment_2(testfile,conditional_probabilities,feature_type,classes,vocabulary,
    priors)

print("SUCCESSES EXP1: ", results_exp1)
print("SUCCESSES EXP2: ", results_exp2)
```
Listing 2: Naive Bayes classifier implementation and Experimental setting.

```python
from deep_translator import GoogleTranslator
import random


def preprocess_doc(doc):
    file = open(doc, "r", encoding="utf-8")
    text = file.read()
    file.close()
    chunks = text.split("\n\n")
    chunks = [chunk.replace("\n", " ") for chunk in chunks]
    return chunks


chunks = preprocess_doc("data/test/persuasion.txt")
# keeping only the book parts
del chunks[1071:]
del chunks[:19]

# getting random paragraphs from the book
random.seed(2)
sample = []
for i in range(0, 3):
    idx = random.randrange(0, len(chunks))
    sample.append(chunks[idx])
    ++i

# translating paragraphs and storing them in a dict
evaluation = {}
languages = [["nl"], ["fr"], ["ru"], ["zh-cn"], ["nl", "fr"], ["ru", "zh-cn"]]

for i, el in enumerate(sample):
    evaluation[i] = {"original": el}
    for l in languages:
        for language in l:
            translated = GoogleTranslator(target=language).translate(el)
        translated = GoogleTranslator(target='en').translate(translated)
        key = "_".join(l)
        evaluation[i][key] = translated

# saving in a txt file
file = open("evaluation.txt", "a", encoding="utf-8")
for i, el in evaluation.items():
    file.write("\n" + str(i))
    for lang, text in el.items():
        file.write("\n\t" + lang + " :\n\t" + text)
file.close()
```

Listing 3: Evaluation script.

References

[1] Matthias Hagen, Martin Potthast, and Benno Stein. “Overview of the Author Obfuscation Task at PAN 2017: Safety Evaluation Revisited”. In: *CLEF*. 2017.

[2] Yonghui Wu et al. *Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. arXiv: `1609.08144 [cs.CL]`.