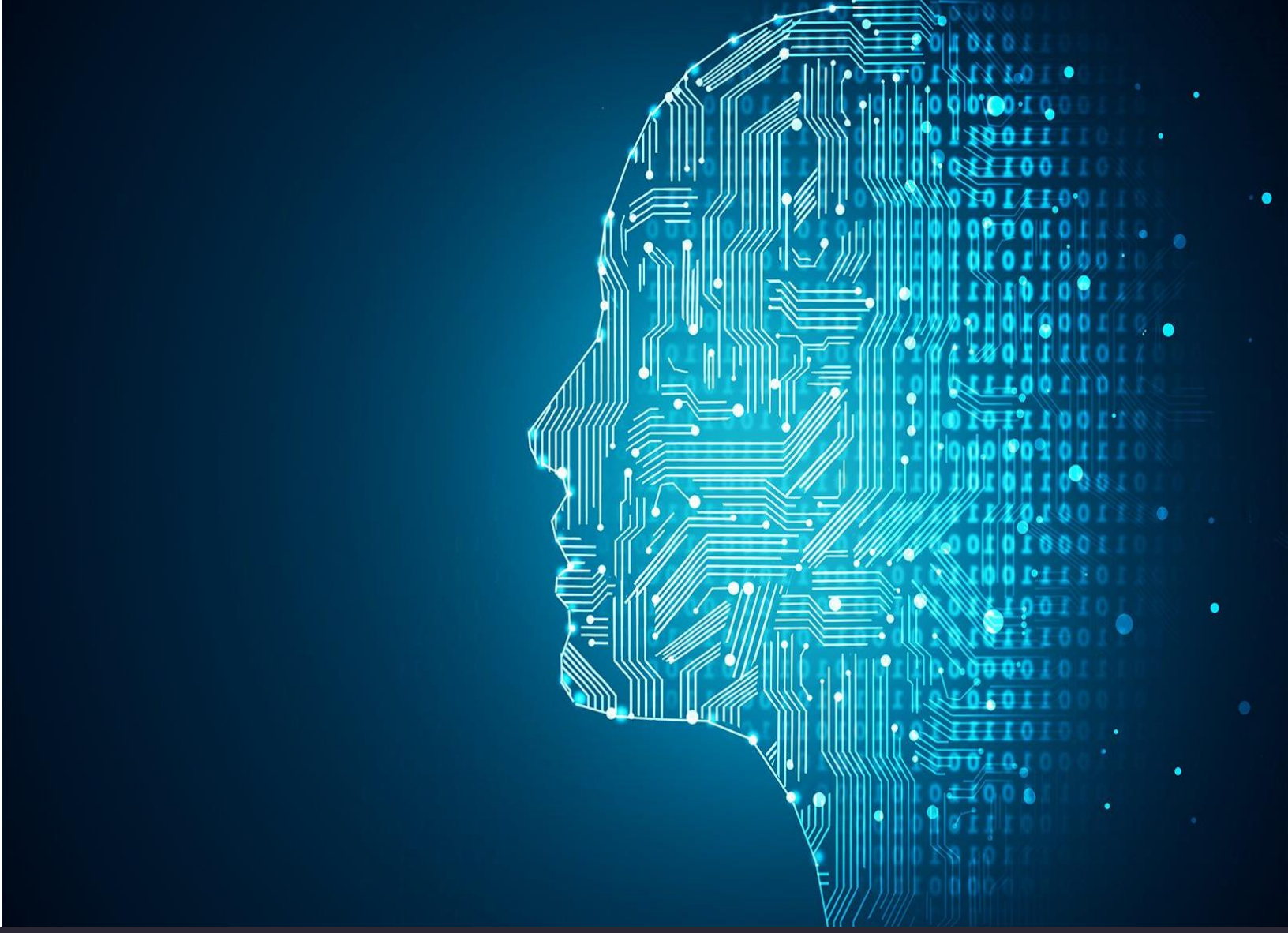

Using machine learning in python to predict an author's gender

As a project for Basic Programming for the premaster CSAI/DSBG, Tilburg University

Group: Warsaw

By Andrew Favia

Gideon Thür

Siebe Albers

Contents

Research question

Can a machine learning algorithm be created with the scikit learn library in python to predict whether a book has been written by a male or a female with an accuracy of 80 percent?

Objectives of the project

The project consisted of mainly three sub objectives, where mainly three libraries had to be utilized:

- Phase 1: Downloading and cleaning textbook files from the Gutenberg project website, by utilizing the FTP library.
- Phase 2: Feeding the cleaned textfile books to a supervised learning, by utilizing the sci-kit learn library.
- Phase 3: Graphically representing the results by utilizing the seaborn library.

Phase 1: FTP-library

Downloading textbook files from the Gutenberg project website, by utilizing the FTPlib library of Python.

In order to download books in .txt formats, we utilized the FTP facilitation offered by the following website: https://www.mirrorservice.org/sites/gutenberg.org/, The Gutenberg project is a "a volunteer effort to digitize and archive cultural works, to "encourage the creation and distribution of eBooks". Therefore, the acquisition of the textfiles was legal.

We noticed that there were tens of thousands of ebooks availiable. These ebooks are located in specific directories (figure 1), in the form of txt.files. We began by creating a script, utilizing the FTP functions to connect to the ftp server, cycle through the directories, and download specific txt. Files (see appendix 1 for the specific script). There were difficulties at first, because the digger() and worm() function that we created (appenix 1) would stop in certain directories. For instance, it would do directories 1/1/1/1/9999 and all those in between, but once it had to go to /2 and do /2/2/2 etc., it would just stop. Conquering problems like these were mainly done by utilizing online sources and trial and error approaches. Eventually, we produced functions that could dig into all directories and download the textfiles, the end product of that script can be found in appendix 2.

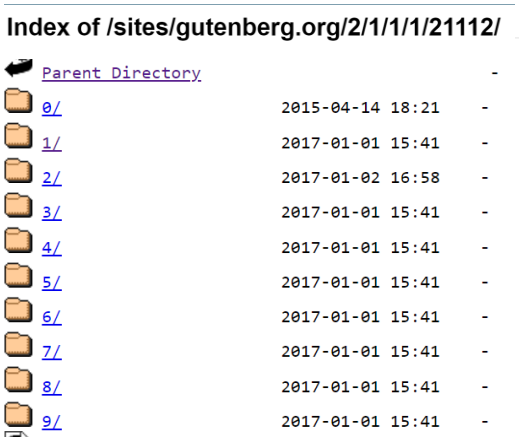

Figure 1, The directories on the Gutenberg FTP website

cleaning and organizing textbook files from the Gutenberg project website, by utilizing regular expressions

The first objective in this phase was to create a function [name_builder()] that would return the title and name from the textfile; if the title or author was missing, we deleted it.

We also had to identify the non English books by making use of RE expressions, so we could identify and delete them.

Then we used the identified title and author to save the text file under those two variables in either the female or male folder on our operating system; to identify whether the author was male or female, we utilized a comprehensive list of female and males names downloaded from the internet and simply used if statements to see if their was a match between the two.

Addionally, by making use of RE expressions, we removed meta data from the books that could increase the bias in our machine learning algorithm.

Phase 2: Scikit-learn

Aim
We decided to use the scikit-learn library to build a machine that could learn to distinguish between books written by males and females.

Walkthrough
Even before deciding which algorithm to use, we had to organize to properly convert the text data into numerical form. In order to do this the bag-of-words model was applied. This model consists in the creation of a matrix (or set of numpy arrays) where each row represents a book, in this case, and each column a word of all the words present in all of the books. It is already clear that the resulting matrix, especially when using novels as data, will be a humongous sparse matrix. This may not be a problem for powerful systems, however, we wanted to be able to execute this experiment on most modern laptops, and in order to do that some further data cleaning to the word-level presented as necessary.

First step was to use the CountVectorizer from the library, using it along with a snowball stemmer to clean each book of stop words (words which are not useful and very repetitive such as: but, yes, or, could, etc...) and then delete suffixes and prefixes from words, therefore reducing the set of words present in each book since, by way of example, instances of "wanted", "wants" and "want" are all reduced to their root "want". Moreover, the vectorizer will take in consideration not only single words but also bigrams, trigrams and 4-grams (set of phrases composed by 2,3 and 4 words).

Next step was to divide the books based on author's gender and give category tags to each book (namely "M" for males and "W" for females). Once that was done, the books were shuffled and then divided into a training set (about 2/3rds of the dataset) and a testing set (about 1/3 of the dataset). The training set will be fed to the machine in order to teach it what may distinguish a book written by a female from a book written by a male and viceversa. The testing set will be used to test the accuracy of the trained algorithm, it will be provided the books but not the category, the machine will give its guess and then it will be compared with the actually known category.

Before the training, however, the dataset converted into an array has to be fit by using a TFIDF (term frequency–inverse document frequency) algorithm. This is a way of calculating word frequencies for texts and it is usually always applied in the bag-of-words model.

Finally, we transformed the resulting TFIDF sparse matrix into a dense one (we want to save resources and make this machine learning test feasable for machines with just 8gb of RAM) and feed it to the LinearSVC function which will apply a linear algorithm to the training set.

Challenges

The main difficulties of the project, in general, were found in the extraction and cleaning of data, as previously mentioned. However, this section as well had its own challenges that will be explained. Furthermore, finding the right algorithm, making the project scalable to most modern PCs and having all the other modules created fit together nicely. In order to find the right algorithm, different ones were tested (Gaussian, Bernoulli, KNN and LinearSVC) and only the best performing one was selected. Same method was applied in the choice of n-grams to include.

Results

At the end of the training, with the data collected by us (500 books, 250 by each gender), the machine has given average accuracies of 71%, 74%, 76%, 79%, 72%, 78%, 80%, 72%, 74% and 75% after 10 different experiments. All the results are clearly above chance, although probably not good enough for production standards. Therefore, it can be concluded that there are differences in the writing styles, namely the choice of words or word ordering, of men and women.

Limitations

It must be noted that, although the results are promising, the machine may be slightly biased by some of the metadata that were difficult to remove in the previous section, and therefore the accuracy may increase or decrease if that information was to be completely removed. However, given the scope of the project, we are satisfied with our results and the possible improvement that may be brought in the future.

Phase 3: Seaborn-library

The objective of the third phase was to Graphically represent the results that came together in second phase
. This objective was realized by utilizing the Seaborn library. Initially, we tried using the Matplotlib library. However, after some investigation we found that Seabourn gives us nicer graphs in less lines of code.
The main issues in representing the distributions of our books was getting the grid and scaling of the plots right. In the end we concluded that the function distplot would display the most comprehensive layout. The end result of the script can be seen in appendix 5.

Appendixes

Phase 1 : FTP-library

Appendix 1: First version of the FTP script

```
1.  from ftplib import FTP
2.
3.  ftp = FTP('ftp.ibiblio.org')
4.
5.  ftp.login()
6.
7.  ftp.cwd("pub/docs/books/gutenberg/") #change working directory
8.
9.  out_file = open("C:/Users/Sa/desktop/writingdirs.txt", "w")
10.
11. print(ftp.nlst())
12.
13. # ================================================================================
14. #
15. # ================================================================================
16. def txtdownload():
17.     for _file in ftp.nlst():
18.         if _file.endswith(".txt"):
19.             ftp.retrbinary("RETR") #retrieve a copy of the file
20.
21.
22. def getdirs1():
23.     list_dirs = []
24.     for _dir in ftp.nlst():
25.         if str(_dir).isdigit():
26.             directory = "pub/docs/books/gutenberg/" + str(_dir)
27.             list_dirs.append(directory)
28.
29.     return list_dirs
30.
31. def getdirs2():
32.     list_dirs = []
33.     for _dir in ftp.nlst():
34.         if str(_dir).isdigit():
35.             list_dirs.append(_dir)
36.
37.     return list_dirs
38.
39. # ================================================================================
40. #
41. # ================================================================================
42. def digger():
43. # THIS FUNCTION GETS A LIST OF THE FIRST 2 LAYERS OF DIRECTORIES
44.     counter = 0
45.     list_of_dirs = []
46.     second_list = []
47.     #uses the getdirs2 function to get the first layer
48.     list_of_dirs.extend(getdirs2())
49.
50.     print("LIST: ", list_of_dirs)
```

```
51.
52.     if len(list_of_dirs) > 0:
53.         #first for loop gets the second layer
54.         for directory in list_of_dirs:
55.
56.             ftp.cwd(directory) #cwd = change working directory
57.
58.             second_list.append(ftp.pwd())
59.             for _dir in getdirs2():
60.
61.                 final = str(ftp.pwd()) + "/" + str(_dir)
62.
63.                 second_list.append(final)
64.
65.
66.             counter += 1
67.             ftp.cwd("..")
68.
69.     #once the first two layers are done the worm function starts to dig
70.     lista = worm(second_list)
71.
72.
73.     return lista
74.
75. # ================================================================================
76. #
77. # ================================================================================
78. def worm(_list):
79.     #this function is supposed to dig down all the provided by digger
80.     _list = _list
81.     idx = 0
82. #    idx = len(_list)
83.     final_list = []
84.     lenlist = len(_list)
85.
86.     counter = 0
87.     extra = 0
88.     print("You are in WORM now")
89.
90.     while counter < 2:
91. #        _list = list(set(_list))
92.         try:
93.             if len(_list) > 1:
94.                 print("CICLE HEREEEE")
95.                 counter += 1
96.                 print(len(_list[-idx:]))
97.                 for directory in _list:
98.                     #go in the X directory in the list
99.                     ftp.cwd(directory) #cwd = change working directory
100.
101.                             for _dir in getdirs2():
102.                                 # for each directory in this new directory
103.                                 # get the name of the new dirs and add them to the
104.                                 # next ones to dig into
105.                                 final = str(ftp.pwd()) + "/" + str(_dir)
106.
107.                                 print(final) #THSI IS THE PRINTED DIR
108.                                 out_file.writelines(final + "\n")
109.                                 if final not in final_list:
110.                                     # if these dirs are not in the list of dirs to dig,
    add them
```

```
                                    # otherwise skip them
                                    final_list.append(final)
                                    _list.append(final)
                                    extra += 1
                                # go back to the previous dir in order to start again
                                ftp.cwd("..")
                            idx = extra

                        else:
                            break
                    except:
                        continue

            return final_list
            out_file.close()


        trial = digger()


        print(trial)
```

Appendix 2: final version of the FTP script

```
import matplotlib
import random
from ftplib import FTP

#
================================================================================
# Define functions
#
================================================================================

def handleDownload(block, fileToWrite):
    # callback handler for the ftp.retrbinary function
    fileToWrite.write(block)


def txtdownload():
    for _file in ftp.nlst():
        # for each txt file in the directory, create a file with the same
name and copy the contents locally.
        if _file.endswith(".txt"):
            fileToWrite = open(_file, "wb")
            ftp.retrbinary("RETR %s" % _file, fileToWrite.write) #retrieve a
copy of the file
            print("Writing: %s" % _file)
            fileToWrite.close()

def getdirs():
    list_dirs = []
    for _dir in ftp.nlst():
        # for all files in the directory, if that whole filename can be
converted to a digit, it is a dir.
        if str(_dir).isdigit():
            list_dirs.append(_dir)

    return list_dirs

#
================================================================================
# this first section establishes the connection with the Gutenberg's ftp
server
#
================================================================================
```

```
print("Connecting...")

ftp = FTP('ftp.ibiblio.org')

ftp.login()

print("Connected")

ftp.cwd("pub/docs/books/gutenberg/") #change working directory

home = "/pub/docs/books/gutenberg/"

print("Home directory now")


out_file = open("writingdirs.txt", "w") # File where to save the directories
for later use.

#
================================================================================
#
#
================================================================================

dirs = []

for x in range(1,6):
    # Creates the list of base directories in the ftp Gutenberg server to
then dig into.
    # range can go from 1 to 10 in all these 3 layers
    for y in range(0,10):
        for z in range(0,10):
            newdir = "/pub/docs/books/gutenberg/%s/%s/%s" % (x,y,z)
            dirs.append(newdir)

for _dir in dirs:
    # for each directory in the ones listed, if it exists, go into it, go one
level deeper
    try:
        ftp.cwd(_dir)
        dirs2 = getdirs()
        for _dir2 in dirs2:
            try:
```

```
                #if these directories exist, write them in the directory list
txt file
                print(_dir2)
                final = str(ftp.pwd()) + "/" + str(_dir2)
                print(final)
                out_file.writelines(final + "\n")
            except:
                continue
        ftp.cwd(home)
    except:
        continue

# close the directory list txt file
out_file.close()

print("Got all directories")
print("Starting to find and download books now...\n")

with open("writingdirs.txt", "r") as _dirs:
    _dirs = _dirs.readlines()

for _dir in _dirs:
    #for each directory saved in the txt file, go in that directory and
download all the txt files.
    try:
        _dir = _dir.rstrip()
        ftp.cwd(_dir)
        print(ftp.pwd())
        txtdownload()
        ftp.cwd(home)
    except:
        continue
```

appendix 3: cleaning the files

```
import
re
        import os
        import filecleaner

        # ================================================================================
        # Define functions
        # ================================================================================

        def name_builder(_text):

            # find the title and author line and return them.
            title = re.search(r"Title:(.+)",_text).group(1)
            author = re.search(r"Author:(.+)",_text).group(1)

            print(title)
            print(author)
            return title,author

        # ================================================================================
        #
        # ================================================================================

        # Create the folders for man and women authors if they are not there.
        if "Men" not in os.listdir():
            os.mkdir("Men")
        if "Women" not in os.listdir():
            os.mkdir("Women")

        in_files = os.listdir()
        for _file in in_files:
            # find all the books, consider them only if they are in English
            if _file.endswith(".txt"):

                text= open(_file , "r",encoding="utf-8",errors="ignore")
                text=text.read()
                if "Language: English" not in text:
                    print(_file)
                    in_files.remove(_file)

        for _file in in_files:
```

```
    if _file.endswith(".txt"):
        # of all the English books, consider only them that provide title and
author formatted this way.
        text= open(_file , "r",encoding="utf-8",errors="ignore")
        text=text.read()
        if "Author:" not in text or "Title:" not in text:
            print(_file)
            in_files.remove(_file)

# ================================================================================
#
# ================================================================================

for _file in in_files:

    if _file.endswith(".txt"):
        # Rename each file that made the selection as "author-book-title.txt"
        text= open(_file , "r",encoding="utf-8",errors="ignore")
        text= text.read()
        try:
            title, author = name_builder(text)
        except:
            continue

        file_name = author + title
        file_name = file_name.lstrip()
        file_name = re.sub("[\. ,\r]+", "-", file_name)
        file_name = re.sub("[\. ,\s\r]+", "-", file_name)
        file_name = file_name + ".txt"
        print(file_name)

        try:
            os.rename(_file ,file_name)
        except:
            print(_file , "this is a double")
            continue

# ================================================================================
# Put books in the Men or Women folder based on author's first name
# ================================================================================

#these are the txt files containing lists of first names.
with open("male_names.txt", "r") as malenames:
```

```
    malenames = malenames.readlines()

malenames = [name.lower().rstrip() for name in malenames]

with open("female_names.txt", "r") as femalenames:
    femalenames = femalenames.readlines()

femalenames = [name.lower().rstrip() for name in femalenames]

nono_files = ["male_names.txt","female_names.txt"]


for _file in os.listdir():
    # for each txt file in the folder, check if the first name is in either name
list
    # if so, move the file into the corresponding dir.
    if _file.endswith(".txt") and _file not in nono_files:
        index = _file.find("-")
        print(_file[:index])
        if _file[:index].lower() in femalenames:
            print("Female! ", _file[:index])
            os.rename(_file,"./Women/"+_file)

        elif _file[:index].lower() in malenames:
            print("Male!", _file[:index])
            os.rename(_file,"./Men/"+_file)


os.chdir("Women")

# use the file cleaner to delete the first and last part of the books which
usually contain metadata and info
# that might create bias in the machine.

for book in os.listdir():
    try:
        print(book)
        filecleaner.file_cleaner(book)
    except:
        continue

os.chdir("..")
```

```
os.chdir("Men")

for book in os.listdir():
    try:
        print(book)

        filecleaner.file_cleaner(book)
    except:
        continue

os.chdir("..")
```

Phase 2 : Scikit-learn

Appendix 4: Final version of the sci-kit learning script

```
from
scipy
import
stats
        from sklearn.feature_extraction.text import CountVectorizer
        import numpy as np
        import pickle
        import os
        import random
        from matplotlib import pyplot as plt
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.svm import SVC, LinearSVC
        from sklearn.feature_extraction.text import CountVectorizer
        from nltk.stem.snowball import EnglishStemmer

        import graphs

        def stemmed_words(doc):
            #stems words (it gets only the "root" of a word, deleting suffixes). Improves
        accuracy and speed.
            return (stemmer.stem(w) for w in analyzer(doc))

        # Make the directory Graphs if it is not there.
        if "Graphs" not in os.listdir():
            os.mkdir("Graphs")

        # Define what stemmer will be used and which function will turn words into
        numbers.

        stemmer = EnglishStemmer()
        analyzer = CountVectorizer().build_analyzer()

        # Make the directory Women if it is not there.
        if "Women" not in os.listdir():
            os.mkdir("Women")

        women_list = []
        os.chdir("Women")
```

```
# open all the books in the directory and save the content in the proper list.
for _file in os.listdir():
    try:
        with open(_file, "r") as read_file:
            women_list.append(read_file.read())
    except:
        continue

# Give the category tag to each book
women_list = [("W", x) for x in women_list]
# Take only the first 250 books.
women_list = women_list[:250]

os.chdir("..")


men_list = []

# Make the directory Men if it is not there.
if "Men" not in os.listdir():
    os.mkdir("Men")

os.chdir("Men")

# open all the books in the directory and save the content in the proper list.
for _file in os.listdir():
    try:
        with open(_file, "r") as read_file:
            men_list.append(read_file.read())
    except:
        continue


os.chdir("..")

# Give a category tag to each book in this list
men_list = [("M", x) for x in men_list]
# Take only the first 250 books
men_list = men_list[:250]

# Put together the 250 books by men and the 250 books by women.
both_list = women_list + men_list
```

```
# make a word count to vector object which takes single words, bigrams, trigrams
and 4-grams,
# deletes stopwords and stems the remaining
cv = CountVectorizer(ngram_range = (1,4),
                     stop_words = "english",
                     analyzer = stemmed_words)

######
#
# PLOTTING OF DISTRIBUTIONS HERE
# This takes time.
"""
dist_number = 0
for book in both_list:
    if len(book[1]) > 1000:
        try:
            f=plt.figure(figsize=(14,14))
            book_vec = cv.fit_transform([book[1]])
            book_vec = book_vec.toarray()
            print(book_vec.shape)
            graphs.graphics(book_vec)
            plt.savefig("./Graphs/distribution-%s.png" % dist_number)
            plt.close()
            dist_number += 1
        except:
            continue
"""
#######

# The scores of each time you execute the experiment will be stored here.
scoresLinear = []

for testing in range(10):

    # make the selection random.
    random.shuffle(both_list)

    # get the books contents, not their category
    texts = [book[1] for book in both_list]
    print("Number of Books:", len(both_list))

    # get the cateogries for each book (either "Man" or "Woman")
    categories = [x[0] for x in both_list]
```

```
print("Categories: ", set(categories))

# choose the training set size
texts_train = texts[:350]
categories_train = categories[:350]

# choose the testing set size
texts_test = texts[350:]
categories_test = categories[350:]

print("Fitting the Data...\n")
# fit the training set set.
cv_fit=  cv.fit_transform(texts_train)
cv_trans = cv.transform(texts_train)

# get the names of the features.
names = cv.get_feature_names()

# convert the data into a matrix (np array)
arr = cv_fit.toarray()

# apparently in bag-of-word methodology this algorithm is often applied
# Term-Frequency-Inverse-Document-Frequency.
transformer = TfidfTransformer().fit(cv_fit)

# Turn the huge sparse matrix to a dense one to save resources.
x_train_tf = transformer.transform(cv_fit).todense()

print("Training the algorithm...\n")
# Feed the matrix to the LinearSVC algorithm.
linear = LinearSVC().fit(x_train_tf, categories_train)

# Transform the testing set as well now
X_new_counts = cv.transform(texts_test)
X_new_tfidf = transformer.transform(X_new_counts)

# Use the LinearSVC trained algorithm on the testing set
predicted = linear.predict(X_new_tfidf)

# get the average of accuracy and append it to the list of scores.
print("linear: ", np.mean(predicted == categories_test))
scoresLinear.append(np.mean(predicted == categories_test))
```

```
# print the scores, save them in a pickle file.
print(scoresLinear)
pickle.dump(scoresLinear, open( "scoresLinear.p", "wb" ))
```

Phase 3: Seaborn

Appendix 5: Graphically representing the results by utilizing the seaborn library

```
import
matplotlib.pyplot
as plt
                    import numpy as np
                    import seaborn as sns


                    def graphics (randmatrix):

                        import random

                        num_rows, num_columns = randmatrix.shape

                        cols = ["r", "w", "blue", "g", "m", "c"]
                        sns.set_style("darkgrid", {"axes.facecolor": ".1", "grid.color":
                    ".8"})
                        sns.set_context("poster")

                        print(num_rows)
                        sns.distplot(randmatrix[0,], hist=False, color=random.choice(cols))
                        plt.title("Frequency of Words")
                        plt.xlabel("Words")
                        plt.ylabel("Frequencies")
```