МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кубанский государственный университет»
(ФГБОУ ВО «КубГУ»)

Кафедра вычислительных технологий

ОТЧЁТ

по практическому заданию №3

Дисциплина: Методы извлечения информации из сетевых источников

Выполнил:

Студенты 65 группы

Хаммуд Али .

Преподаватель:

Приходько Т. А.

Краснодар

2020

Задание :

Классификация статей Википедии в одну из категорий "наука" или "искусство".

1- Объяснение теории :

В машинном обучении , наивные классификаторы Байеса представляют собой семейство простых «вероятностных классификаторов» на основе применения теоремы Байеса с сильными (наивными) независимостью допущениями между функциями.

Наивный байесовский были изучены с 1960 года . Она была введена (хотя и не под этим именем) в поисковой текст сообщества в начале 1960 - х годов, и остается метод популярным (базовый уровень) для категоризации текста , проблема судейства документов, принадлежащих к одной категории или другой (например , как спам или законный , спорт или политика и т.д.) с частотами слов как особенность. При соответствующей предварительной обработке, он является конкурентоспособным в этой области с более продвинутыми методами, включая машины опорных векторов . Он также находит применение в автоматической медицинской диагностике .

1.1- Вступление

Наивный байесовский простой метод для построения классификаторов: модели, которые присваивают класс метки для проблемных экземпляров, представленных в виде векторов художественных ценностей, где метки класса нарисованы из некоторого конечного множества. Существует не один алгоритм для обучения таких классификаторов, но семейство алгоритмов , основанных на общем принципе: все наивные классификаторы Байеса предположить , что значение конкретной функции является независимым от стоимости любого

другого признака, учитывая переменную класса. Например, плод может рассматриваться как яблоко , если это красное, круглое, и около 10 см в диаметре. Наивный байесовский классификатор считает каждый из этих признаков , чтобы внести свой вклад , независимо от вероятности того, что этот фрукт является яблоко, независимо от каких - либо возможных корреляций между особенностями цветовой, округлости и диаметра.

1.2- Вероятностная модель :

Отвлеченно, наивным Байес является условной вероятностью , модель: дан экземпляр проблемы быть классифицировано, представляется вектором, представляющий некоторые *n* функции (независимые переменные), он присваивает этот экземпляр вероятности $\mathbf{x} = (x_1, \ldots, x_n)$

$$p(C_k \mid x_1, \ldots, x_n)$$

для каждого из *K* возможных исходов или *классов* C_k

Проблема с приведенной выше формулировкой является то , что , если число функций *n* велико или если функция может принимать большое количество значений, то основываясь такой модель на вероятностных таблицах является недопустимой. Поэтому мы переформулировать модель , чтобы сделать его более сговорчивым. Используя теорему Байеса , условная вероятность может быть разложена

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

На простом английском языке, используя байесовскую вероятностную терминологию, вышеприведенное уравнение можно записать в виде

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

На практике существует интерес только в числителе этой дроби, потому что знаменатель не зависит от и значения функций даны, так что знаменатель эффективно постоянная. Числитель эквивалентен совместной вероятности модели C_{xi}

$$p(C_k, x_1, \ldots, x_n)$$

которое можно переписать следующим образом , используя правило цепи для повторных применений определения условной вероятности :

$$\begin{aligned} p(C_k, x_1, \ldots, x_n) &= p(x_1, \ldots, x_n, C_k) \\ &= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2, \ldots, x_n, C_k) \\ &= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2 \mid x_3, \ldots, x_n, C_k) p(x_3, \ldots, x_n, C_k) \\ &= \ldots \\ &= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2 \mid x_3, \ldots, x_n, C_k) \ldots p(x_{n-1} \mid x_n, C_k) p(x_n \mid C_k) p(C_k) \end{aligned}$$

Теперь «наивное» условная независимость допущение вступает в игру: предположат , что каждая функция является условно независимой от любой другой функции для , учитывая категорию . Это означает, что $x_i x_j j \neq i C_k$

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k) \,,$$

Таким образом, совместная модель может быть выражена как

$$\begin{aligned} p(C_k \mid x_1, \ldots, x_n) &\propto p(C_k, x_1, \ldots, x_n) \\ &= p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k)\, \cdots \\ &= p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,, \end{aligned}$$

где обозначает пропорциональность $\propto$

Это означает , что при сделанных предположениях независимости, условное распределение по переменному классу является: ***C***

$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

где доказательство является коэффициент масштабирования зависит только от , то есть константа , если значения переменных художественного известны.

$$Z = p(\mathbf{x}) = \sum_{k} p(C_k)\, p(\mathbf{x} \mid C_k) x_1, \ldots, x_n$$

1.2.1- Построение классификаторов из вероятностной модели

До сих пор обсуждение вывело модель независимой функций, то есть, наивные байесовские вероятностные модели. Наивный байесовский классификатор объединяет эту модель с правилом решения . Одно общее правило заключается в выборе гипотезы , что является наиболее вероятным; это известно как *максимум апостериорной* или *MAP* правило принятия решения. Соответствующий классификатор, классификатор Байеса , это функция , которая назначает метку класса для некоторых *к* следующей: $\hat{y} = C_k$

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

Полиномиальной наивный байесовский

С полиномиальной моделью событий, образцы (векторы признаков) представляют частоты , с которой определенные события были , порожденные мультиномиальной , где есть вероятность того, что событие *я* происхожу (или *K* таких многочлены в мультиклассирует случае). Вектор особенностью является то гистограмма с подсчета количества раз ,

событие *я* наблюдалась в конкретном случае. Это модель событий , как правило , используется для классификации документов, с событиями , представляющими возникновение слова в одном документе (мешок слов предположения). Вероятность наблюдения гистограммы **x** задается

$(p_1, \ldots, p_n)p_i \qquad \mathbf{x} = (x_1, \ldots, x_n)x_i$

$\mathbf{x} = (x_1, \ldots, x_n)x_i$

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

Мультиномиальный наивный байесовский классификатор становится линейным классификатором , выраженные в логе-пространстве:

$$\begin{aligned} \log p(C_k \mid \mathbf{x}) &\propto \log\left(p(C_k) \prod_{i=1}^{n} {p_{ki}}^{x_i}\right) \\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki} \\ &= b + \mathbf{w}_k^\top \mathbf{x} \end{aligned}$$

где и . $b = \log p(C_k) w_{ki} = \log p_{ki}$

2- Объяснение кода :

1- Получение, обработка и архивирование информации.

- Мы пишем url статьи в двух списках «Наука и искусство». Можем добавить любую статью в список.
- Программа будет читать и обрабатывать статьи с помощью библиотеки requests BeautifulSoap.
- Мы используем обработку естественного языка для подготовки текста к классификации.
- Мы храним текст статей в файле CSV с помощью библиотеки pandas.

```
#Мы используем модуль requests для подключения к Интернету и запрашиваем URL статей
#Мы используем BeautifulSoup для чтения данных, которые возвращаются из запроса в
#формате HTML
#Мы используем pandas для упорядочивания информации и сохранения ее в файл CSV
import requests
from bs4 import BeautifulSoup
import pandas as pd
from clear_data_fun import clear_data

def bring_data():
    #cat список имеет категорию, а два списка содержат URL для каждой категории,
    # Мы можем добавить любое количество URL
    cat = ['Art', 'Science']
    #URL искусств
    url_cat1= [ 'https://en.wikipedia.org/wiki/Art',
                'https://en.wikipedia.org/wiki/The_arts',
                'https://en.wikipedia.org/wiki/Drawing',
                'https://en.wikipedia.org/wiki/Graphic_design',
                'https://en.wikipedia.org/wiki/Paint',
                'https://en.wikipedia.org/wiki/Song',
                'https://en.wikipedia.org/wiki/Singing',
                'https://en.wikipedia.org/wiki/Music',
                'https://en.wikipedia.org/wiki/Rhythm',
                'https://en.wikipedia.org/wiki/Artist',
                'https://en.wikipedia.org/wiki/Disco',
                'https://en.wikipedia.org/wiki/Hip-hop_dance',
                'https://en.wikipedia.org/wiki/Art_museum',
                'https://en.wikipedia.org/wiki/Moscow_Museum_of_Modern_Art',
                'https://en.wikipedia.org/wiki/Hand-colouring_of_photographs',
                'https://en.wikipedia.org/wiki/Performance_art',
```

```
                'https://en.wikipedia.org/wiki/Magic_(illusion)',
                'https://en.wikipedia.org/wiki/Poetry_reading',
                'https://en.wikipedia.org/wiki/Casting_(performing_arts)',
                'https://en.wikipedia.org/wiki/Writing']
    # URL науки
    url_cat2= [ 'https://en.wikipedia.org/wiki/Science',
                'https://en.wikipedia.org/wiki/Mathematics',
                'https://en.wikipedia.org/wiki/Computer_program',
                'https://en.wikipedia.org/wiki/Programmer',
                'https://en.wikipedia.org/wiki/Engineering',
                'https://en.wikipedia.org/wiki/Mechatronics',
                'https://en.wikipedia.org/wiki/Robotics',
                'https://en.wikipedia.org/wiki/Physics',
                'https://en.wikipedia.org/wiki/Algorithm',
                'https://en.wikipedia.org/wiki/Computer_science',
                'https://en.wikipedia.org/wiki/Engineer',
                'https://en.wikipedia.org/wiki/Robot',
                'https://en.wikipedia.org/wiki/Motion_planning',
                'https://en.wikipedia.org/wiki/Mechanical_engineering',
                'https://en.wikipedia.org/wiki/Statistics',
                'https://en.wikipedia.org/wiki/Data_science',
                'https://en.wikipedia.org/wiki/Machine',
                'https://en.wikipedia.org/wiki/Electricity',
                'https://en.wikipedia.org/wiki/Engine',
                'https://en.wikipedia.org/wiki/Applied_mathematics']
    data_pd=[]
    for url in url_cat1 :
        # запросить URL
        text_html = requests.get( url, timeout=5).text
        # сохранить данные в виде HTML-кода, и мы берем только данные в <div> с 'id'
= 'bodyContent'
        soup = BeautifulSoup(text_html,"html5lib").find("div", {"id": "bodyContent"
})
        # извлечь текст из данных
        data = soup.get_text().lower()
        # применить NLP к тексту для подготовки его к классификации
        data_ready = clear_data(data)
        # добавить текст и его категорию в список данных
        data_pd.append([data_ready, cat[0]])

    for url in url_cat2 :
        text_html = requests.get( url, timeout=5).text
        soup = BeautifulSoup(text_html,"html5lib").find("div", {"id": "bodyContent"
})
        data = soup.get_text().lower()
        data_ready = clear_data(data)
        data_pd.append([data_ready, cat[1]])
```

```
    # Создать pandas DataFrame
    df = pd.DataFrame(data_pd, columns = ['the_article', 'Class'])
    # установить числовую метку для категории
    df['label'] = df['Class'].apply(lambda x: 0 if x=='Art' else 1)
    # Сохранить данные в файл CSV
    df.to_csv('DATA.csv')
```

1.1- Объяснение функции clear_data() из файла clear_data_fun:

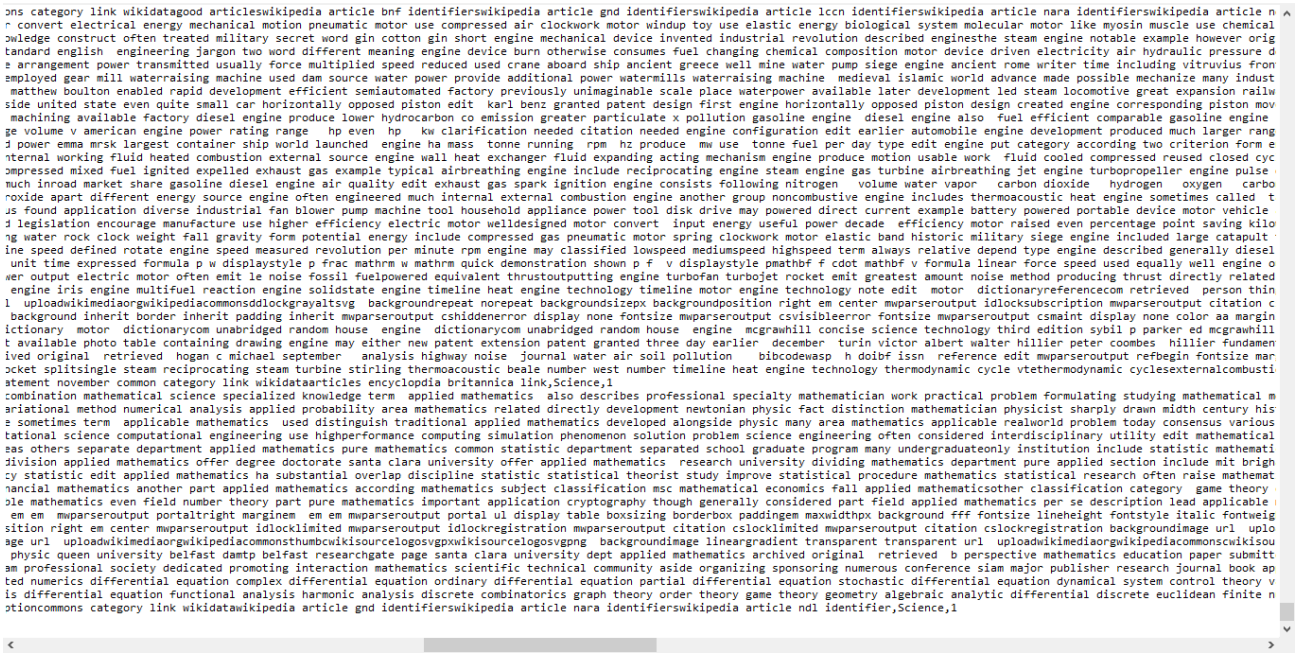
Во первых я воспользовался стандартной функцией из хорошо известной python библиотеки nltk word_tokenize(). Из-за этого возникают некоторые неточности, которые мы обработаем дальше. К тому же эта функция не избавляет нас от знаков препинания. На следующем этапе мы избавляемся от знаков препинания, проверив каждое слово на string.punctuation, Для большей точности я использовал библиотеку re для удаления любой гарнитуры, кроме букв.

На следующем этапе импортировать стоп слова из всё той же библиотеки для работы с естественным языком NLTK и исключить их из нашего конечного списка. Более того, для своих нужд я расширил этот список словами, которые не нужны в конечном итоге.В результате получаем список слов для дальнейшей обработки (рис.1).

```
import re
import nltk
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

def clear_data(text_procssing):
    # применить токенизацию nltk
    tokens = nltk.word_tokenize(text_procssing)
    # удалить знаки препинания
    tokens = [i for i in tokens if ( i not in string.punctuation)]
    for i in range(0, len(tokens)):
# 'Sub' в функции обозначает SubString, в заданной строке выполняется поиск
#определенного шаблона регулярного выражения (3-й параметр), а после нахождения
#шаблона подстроки заменяется на (2-й параметр).
```

```
        tokens[i] = re.sub('[^a-zA-Z]', '', tokens[i])
    # применить лемматизацию nltk
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(w) for w in tokens]
    # удалить стоп-слова
    stop_words = stopwords.words('english')
    stop_words.extend(['encyclopedia', 'wikipedia', 'free', 'wa', 'parser', 'jump',
'navigation', 'jump', 'search'])
    tokens = [i for i in tokens if ( i not in stop_words )]
    text_ready = ' '.join(tokens)
    return text_ready
```



(рис.1) Готовая информация

2- Процесс классификации с помощью библиотеки scikit-learn:

- Деление статьи на учебную часть и тестовую часть.
- Векторизация нашего набора данных
- Построение и оценка модели

```
#pyplot в основном предназначен для интерактивных графиков и простых случаев
программной генерации графиков
#Seaborn - это библиотека визуализации данных Python, основанная на matplotlib.
# Она предоставляет высокоуровневый интерфейс для рисования привлекательной и
информативной статистической графики.
# мы используем его для рисования тепловой карты результата
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

def classifier_MultinomialNB():
    df=pd.read_csv("DATA.csv", usecols=['the_article', 'Class', 'label'])

    from sklearn.model_selection import train_test_split
    # разделить данные и цели на наборы обучения и тестирования
    X_train, X_test, y_train, y_test = train_test_split(df['the_article'], df['labe
l'], test_size=0.2)

    from sklearn.feature_extraction.text import CountVectorizer
    # создать экземпляр векторизатора
    cv = CountVectorizer()
    # подгонка и преобразование данных обучения в матрицу термина документа
    X_train_cv = cv.fit_transform(X_train)
    # преобразовать данные тестирования в матрицу термина документа
    X_test_cv = cv.transform(X_test)

    from sklearn.naive_bayes import MultinomialNB
    # инстанцировать полиномиальную наивную байесовскую модель
    naive_bayes = MultinomialNB()
    # Подходить наивный байесовский классификатор в соответствии с X_train_cv,
y_train
    naive_bayes.fit(X_train_cv, y_train)
    # Выполнить классификацию по массиву тестовых векторов X_test Cv.
    predictions = naive_bayes.predict(X_test_cv)

    from sklearn.metrics import accuracy_score, precision_score, recall_score
    # рассчитать точность прогнозов
    accuracy_value=accuracy_score(y_test, predictions)
##    print('Accuracy score: ', accuracy_score(y_test, predictions))
```

```
##    print('Precision score: ', precision_score(y_test, predictions))
##    print('Recall score: ', recall_score(y_test, predictions))


    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, predictions)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax)
    # ярлыки, заголовки и галочки
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title('The accuracy of this predictions is '+str(accuracy_value));
    ax.xaxis.set_ticklabels(['Art', 'Science']); ax.yaxis.set_ticklabels(['Art', 'S
cience']);
    plt.show()
```

Это файл main.py , Здесь мы запускаем код и GUI (рис.2) с помощью библиотеки tkinter

```
# Импортировать модули и функции
from tkinter import *
from bring_data_from_url import bring_data
from classifier_MultinomialNB_func import classifier_MultinomialNB
from classifier_MultinomialNB_func_iteration import classifier_MultinomialNB_Avg

# создать новое окно и назначить его переменной 'root'
root= Tk()
# установить заголовок окна
root.title("classifier")
# установить значок окна
root.iconbitmap("Img/kubsu.ico")
# установить размеры окна
root.geometry("530x350")
# фиксированный размер окна
root.resizable(0, 0)
# Создать виджета Label с текстом и присвоение его переменной; мы используем виджет
#для отображения текста на экране
Label_Text = Label(root, text='This code will bring the data for 40 url from wikipe
dia.org .\nThey belong to two categories ART and SCIENCE.')
Label_Text.grid( row=0, column=0, padx=5, pady=5)

# Функция для чтения данных из файла
def bring():
    bring_data()
```

```
# Создать виджета кнопки; Кнопка, которая может содержать текст и может выполнять
действие «команда для запуска функции» при нажатии
Button_bring = Button(root, text="Bring the data and save them as DATA.csv", comman
d=bring)

#использование менеджера геометрии .grid() для определения положения любого виджета
#padx добавляет отступ в горизонтальном направлении.
#pady  добавляет отступы в вертикальном
Button_bring.grid(row=1, column=0, padx=5, pady=5)

# Функция запуска полиномиального наивного байесовского классификатора
def classify():
    classifier_MultinomialNB()

Label_classify = Label(root, text='Using Multinomial Naive Bayes The code will choo
se 20%of dataset for testing and 80% to train it.\nThe result will be in confusion
matrix.')
Label_classify.grid( row=2, column=0, padx=5, pady=5)

Button_classify = Button(root, text="Start the classification", command=classify)
Button_classify.grid(row=3, column=0, padx=5, pady=5)

# Функция для вычисления среднего числа итераций запуска классификатора
def get_avg():
    #использовать .get () для извлечения текста и присвоения его переменной
    num = int(iteration.get())
    if num <= 50 and num > 0:
        avg = "{:.2f}".format(classifier_MultinomialNB_Avg(num) * 100)
        Label_iteration = Label(root, text='The avarge is : '+str(avg))
        Label_iteration.grid( row=7, column=0, padx=5, pady=5)

Label_iteration = Label(root, text='The code here will repeat the classification
 operation by the number which you write.\nThe average of accuracy of classifier
 will appear in the bottom.\nHow many times you want to classify? [1:50]')
Label_iteration.grid( row=4, column=0, padx=5, pady=5)

# Виджет ввода текста, который позволяет пользователю писать текст
iteration = Entry(root, width=35, borderwidth=5)
iteration.grid(row=5, column=0, padx=5, pady=5)

Button_iteration = Button(root, text="Get the average", command=get_avg)
Button_iteration.grid(row=6, column=0, padx=5, pady=5)

#window.mainloop () сообщает Python запустить цикл событий Tkinter.
```

```
#Этот метод прослушивает события, такие как нажатия кнопок или нажатия клавиш, и
#блокирует любой код, который следует за ним, до тех пор, пока не будет закрыто
#вызываемое окно.
root.mainloop()
```

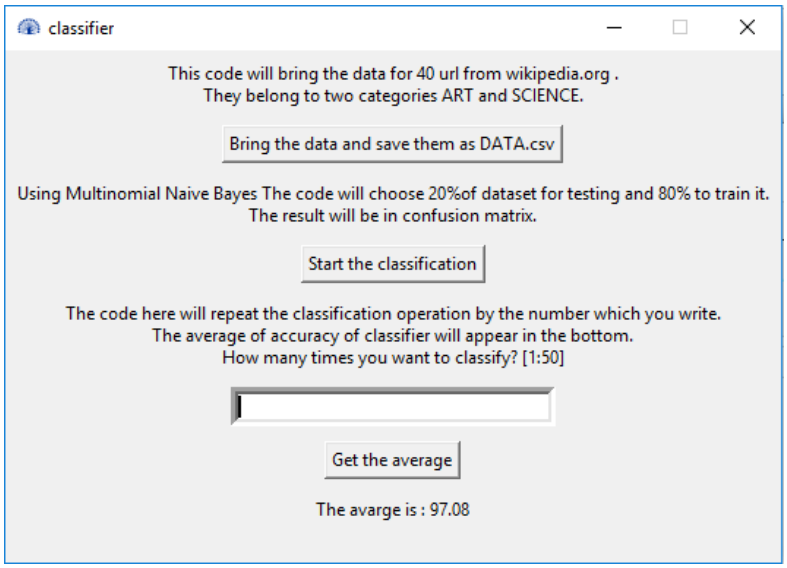
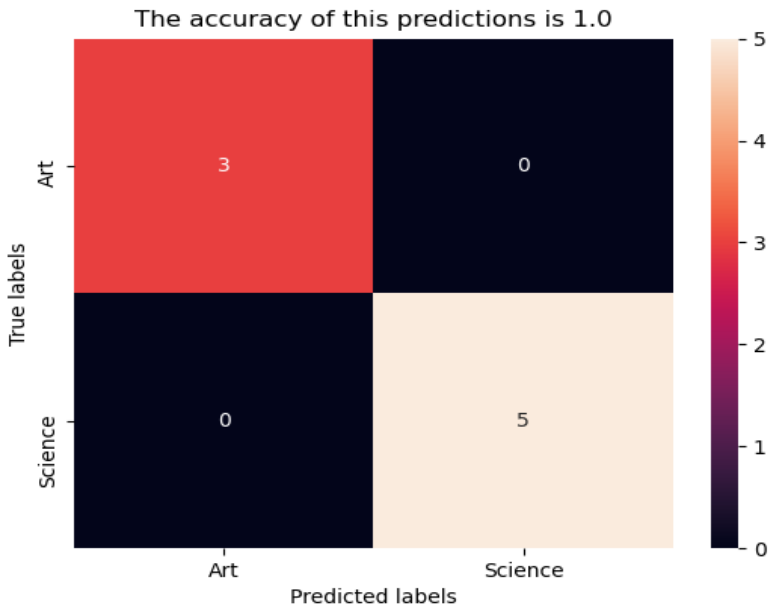


рис.2. GUI



Полученный нами результат показывает, что классификатор правильно предсказал 100 процентов