Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический университет

имени В.Ф.Уткина»

Кафедра «Электронные вычислительные машины»

Отчет по практическому занятию №8

на тему

«Наивный байесовский классификатор»

по дисциплине  
**«Машинное обучение»**

Выполнили:

Студенты группы №140

Бригада 7

Сафаров Д. А.

Тимохин Е. М.

Проверила:  
ас. Панина И.С.

**Цель работы**

Реализовать наивный байесовский классификатор с помощью языка Python.

**Задание**

Реализуйте наивный байесовский классификатор, предназначенный для проверки на спам-сообщение, с использованием возможностей библиотеки Scikit-learn. Точность должна превышать 80 %.

**Практическая часть**

Код программы представлен в приложении А. Результат её работы представлен на рисунке 1.

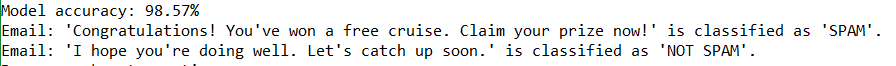


Рисунок 1 – Результат работы нейросети

Приложение А. Текст программы

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import make\_pipeline

from sklearn.model\_selection import train\_test\_split

import requests

from zipfile import ZipFile

from io import BytesIO

# Ссылка на набор данных "Spam SMS Collection" на UCI Machine Learning Repository

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspamcollection.zip"

# Загрузка и распаковка архива с данными

response = requests.get(url)

with ZipFile(BytesIO(response.content)) as zip\_file:

zip\_file.extractall()

# Чтение данных из файла

with open("SMSSpamCollection", "r", encoding="utf-8") as file:

data = file.readlines()

# Разделение текстов и меток

X\_data = [line.strip().split("\t")[1] for line in data]

y\_data = np.array([1 if line.strip().split("\t")[0] == "spam" else 0 for line in data])

# Создание класса SpamAnalyzer

class SpamAnalyzer:

def \_\_init\_\_(self, alpha=1.0):

self.alpha = alpha

self.model = None

def train(self, X\_train, y\_train):

self.model = make\_pipeline(TfidfVectorizer(), MultinomialNB(alpha=self.alpha))

self.model.fit(X\_train, y\_train)

def classify(self, email):

return self.model.predict([email])[0]

def accuracy(self, X\_test, y\_test):

return self.model.score(X\_test, y\_test)

# Разделение на обучающий и тестовый наборы данных

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_data, y\_data, test\_size=0.2, train\_size=0.8, random\_state=42)

# Обучение модели на обучающем наборе данных

analyzer = SpamAnalyzer(alpha=0.1) # Устанавливаем гиперпараметр alpha

analyzer.train(X\_train, y\_train)

# Оценка точности модели на тестовом наборе данных

accuracy = analyzer.accuracy(X\_test, y\_test) \* 100

print("Model accuracy: {:.2f}%".format(accuracy))

# Добавляем два новых письма для проверки работы классификатора

new\_emails = [

"Congratulations! You've won a free cruise. Claim your prize now!",

"I hope you're doing well. Let's catch up soon."

]

# Классификация новых писем и вывод результатов

for email in new\_emails:

classification = "SPAM" if analyzer.classify(email) == 1 else "NOT SPAM"

print("Email: '{}' is classified as '{}'.".format(email, classification))