Министерство науки и высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования «Рыбинский государственный

авиационный технический университет имени П. А. Соловьева»

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И СИСТЕМЫ УПРАВЛЕНИЯ

Математического и программного обеспечения электронных

вычислительных средств

ОТЧЁТ

по дисциплине:

«Методы и алгоритмы анализа данных»

на тему:

«Классификация текстов»

Выполнил: студент группы ИВМ-24 Морозов А. А.

Руководитель: ассистент Вязниковцев Д. А.

Рыбинск 2024

Содержание

[Цель работы 3](#_Toc183812873)

[1 Первичная обработка данных 4](#_Toc183812874)

[2 Удаление стоп-слов и стемминг 5](#_Toc183812875)

[3 «Мешок слов» и обучение моделей 7](#_Toc183812876)

[Вывод 11](#_Toc183812877)

# Цель работы

Целями данной лабораторной работы являются:

- изучение теории о мешке слов;

- первичная обработка данных;

- ознакомительная работа с библиотекой *nltk*;

- обучение моделей *Random Forest* и *Gradient Boosting*.

# 1 Первичная обработка данных

В начале необходимо загрузить 2 датасета с негативными и позитивными комментариями из популярной социальной сети:

import pandas as pd

dfN = pd.read\_csv('negative.csv', header=None, sep=';', quotechar='"')

dfP = pd.read\_csv('positive.csv', header=None, sep=';', quotechar='"')

Теперь конкатенируем оба *dataframe*, перемешиваем данные, добавляем имена для столбцов и выводим в консоль получившийся результат (рисунок 1.1):

df = pd.concat([dfN, dfP])  
df = df.sample(frac=1).reset\_index(drop=True) # перемешали данные  
print(f"df:\n{df}")  
df.columns = ["id", "date", "name", "message", "sentiment"] + ["undefined" for i in range(df.shape[1] - 5)]  
print(f"df:\n{df}")

Изображение выглядит как снимок экрана, текст, Шрифт

Автоматически созданное описание

Рисунок 1.1 – Получившийся *dataframe*

# 2 Удаление стоп-слов и стемминг

Для дальнейшей работы с *dataframe* необходимо удалить стоп-слова и выполнить стемминг оставшихся.

Для удаления стоп-слов воспользуемся функцией *preprocess\_text*:

def preprocess\_text(texts):

stop\_words = set(stopwords.words('russian'))

regex = re.compile('[^а-я А-Я]')

preprocess\_texts = []

for i in tqdm.tqdm(range(len(texts))):

text = texts[i].lower()

text = regex.sub(' ', text)

word\_tokens = word\_tokenize(text, language="russian")

filtered\_sentence = [w for w in word\_tokens if not w in stop\_words]

preprocess\_texts.append(' '.join(filtered\_sentence))

return preprocess\_texts

Для стемминга воспользуемся функцией *stemming\_texts*:

def stemming\_texts(texts):

st = SnowballStemmer("russian")

stem\_text = []

for text in tqdm.tqdm(texts):

word\_tokens = word\_tokenize(text, language="russian")

stem\_text.append(' '.join([st.stem(word) for word in word\_tokens]))

return stem\_text

Выведем полученный *dataframe* в консоль (рисунок 2.1).

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 2.1 – Изменённый *dataframe*

# 3 «Мешок слов» и обучение моделей

Перед обучением моделей необходимо представить текст как вектор – «Мешок слов» и его взвешенную версию с помощью функций *prepare\_data* и *prepare\_data\_TF* соответственно:

def prepare\_data(df, text\_column, target\_column, test\_size=0.3, random\_state=42):

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(df[text\_column])

y = df[target\_column]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

return X\_train, X\_test, y\_train, y\_test

def prepare\_data\_TF(df, text\_column, target\_column, test\_size=0.3, random\_state=42):

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(df[text\_column])

y = df[target\_column]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

return X\_train, X\_test, y\_train, y\_test

Для обучения модели градиентного спуска и модели случайного дерева воспользуемся функцией *learning*:

def learning(X\_train, y\_train, X\_test, y\_test, num, bag):

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

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import accuracy\_score

RFC, GBC = RandomForestClassifier(), GradientBoostingClassifier()

param\_grid\_rf = {'max\_depth': [5, 6, 7], 'max\_features': ['sqrt', 'log2']}

grid\_rf = GridSearchCV(RFC, param\_grid\_rf, cv=3, n\_jobs=-1)

grid\_rf.fit(X\_train, y\_train)

y\_predicted = grid\_rf.predict(X\_test)

accuracy\_score\_n = accuracy\_score(y\_predicted, y\_test)

print(f"Лучшие параметры у дерева: {grid\_rf.best\_params\_}, {num}, {bag}, {accuracy\_score\_n}")

param\_grid\_gb = {'max\_depth': [5, 6, 7], 'max\_features': ['sqrt', 'log2']}

grid\_gb = GridSearchCV(GBC, param\_grid\_gb, cv=3, n\_jobs=-1)

grid\_gb.fit(X\_train, y\_train)

y\_predicted = grid\_gb.predict(X\_test)

accuracy\_score\_n = accuracy\_score(y\_predicted, y\_test)

print(f"Лучшие параметры у градиентного спуска: {grid\_gb.best\_params\_}, {num}, {bag}, {accuracy\_score\_n}")

Используем разные данные для обучения моделей:

- данные до удаления стоп-слов и стемминга;

- данные после удаления стоп слов, но без стемминга;

- данные после удаления стоп-слов и стемминга.

Обучение для разных данных будет производиться в нескольких потоках и результаты сохраняться в файл *output.txt*:

logs = "output.txt"

with open(logs, "w", encoding="utf-8") as file:

# Перенаправляем вывод

sys.stdout = file

X\_train, X\_test, y\_train, y\_test = prepare\_data(df, "message", "sentiment")

X\_train\_preprocess, X\_test\_preprocess, y\_train\_preprocess, y\_test\_preprocess = prepare\_data(preprocess\_data, "message", "sentiment")

X\_train\_stemming, X\_test\_stemming, y\_train\_stemming, y\_test\_stemming = prepare\_data(data\_stemming, "message", "sentiment")

threading.Thread(target=learning, args=(X\_train, y\_train, X\_test, y\_test, "исходные тексты", "Обычный мешок слов")).run()

threading.Thread(target=learning, args=(X\_train\_preprocess, y\_train\_preprocess, X\_test\_preprocess, y\_test\_preprocess, "предварительно обработанные тексты", "Обычный мешок слов")).run()

threading.Thread(target=learning, args=(X\_train\_stemming, y\_train\_stemming, X\_test\_stemming, y\_test\_stemming, "тексты после стемминга", "Обычный мешок слов")).run()

X\_trainTF, X\_testTF, y\_trainTF, y\_testTF = prepare\_data\_TF(df, "message", "sentiment")

X\_train\_preprocessTF, X\_test\_preprocessTF, y\_train\_preprocessTF, y\_test\_preprocessTF = prepare\_data\_TF(preprocess\_data, "message", "sentiment")

X\_train\_stemmingTF, X\_test\_stemmingTF, y\_train\_stemmingTF, y\_test\_stemmingTF = prepare\_data\_TF(data\_stemming, "message", "sentiment")

threading.Thread(target=learning, args=(X\_trainTF, y\_trainTF, X\_testTF, y\_testTF, "исходные тексты", "Взвешенный мешок слов")).run()

threading.Thread(target=learning, args=(X\_train\_preprocessTF, y\_train\_preprocessTF, X\_test\_preprocessTF, y\_test\_preprocessTF, "предварительно обработанные тексты", "Взвешенный мешок слов")).run()

threading.Thread(target=learning, args=(X\_train\_stemmingTF, y\_train\_stemmingTF, X\_test\_stemmingTF, y\_test\_stemmingTF, "тексты после стемминга", "Взвешенный мешок слов")).run()

Содержимое файла *output.txt* представлено на рисунке 3.1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3.1 – Содержимое файла *output.txt*.

# Вывод

Полученные результаты говорят, что обе модели имеют точность более 50%, точность у градиентного спуска выше, чем у случайного дерева. Данные результаты можно объяснить тем, что тип сообщения зависит не только от набора используемых в нём слов, но и от контекста самого сообщения.