Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Рубежный контроль №2

по дисциплине «Методы машинного обучения»

Методы обработки текстов

Вариант 1

ИСПОЛНИТЕЛЬ:

студент ИУ5-23М
Анцифров Н.С.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю. Е.

___ "_____" 2024 г.

Москва, 2024

1 Вариант и задачи

Необходимо решить задачу классификации текстов на основе любого датасета (кроме примера, который рассматривался в лекции). Классификация может быть бинарной или многоклассовой. Целевой признак из датасета может иметь любой физический смысл, примером является задача анализа тональности текста.

Необходимо сформировать два варианта векторизации признаков – на основе CountVectorizer и на основе TfidfVectorizer.

В качестве классификаторов необходимо использовать два классификатора по варианту группы: **LinearSVC** и **LogisticRegression**.

Для каждого метода необходимо оценить качество классификации. Сделать вывод о том, какой вариант векторизации признаков в паре с каким классификатором показал лучшее качество.

2 Описание набора данных

Был выбран набор данных, содержащий данные о спам-рассылках: https://www.kaggle.com/datasets/team-ai/spam-text-message-classification.

Набор данных имеет следующие атрибуты:

– *Category* – категория сообщения (ham / spam);

– *Message* – сам текст сообщения.

Целевой признак – *Category*.

3 Выполнение работы

3.1 Импорт библиотек и загрузка набора данных

Импортируем библиотеки и загрузим данные (см. рисунок 1).

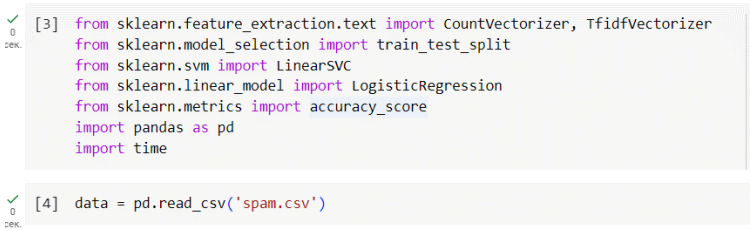

Рисунок 1 – Импорт библиотек и загрузка данных

Выведем информацию о датасете (см. рисунок 2).

```
[9] data.head()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Далее: Посмотреть рекомендованные графики

```
[10] data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  5572 non-null   object
 1   Message   5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

Рисунок 2 – Информация о датасете

3.2 Проверка на наличие пропусков

Проверим данные на наличие пропусков (см. рисунок 3).

```
[11] prop_mask = data.isna()
     props = prop_mask.sum()
     props

Category    0
Message     0
dtype: int64
```

Рисунок 3 – Проверка на пропуски

Пропусков данных нет.

3.3 Разделение выборки на обучающую и тестовую

Разделим выборку на обучающую и тестовую (см. рисунок 4).

```
[13] X, Y = data['Message'], data['Category']
     X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)

     time_arr = []
```

Рисунок 4 – Разделение выборки

3.4 Векторизация признаков

Проведём векторизацию признаков с помощью CountVectorizer (см. рисунок 5).

```
[15] count_vect = CountVectorizer()
     X_train_counts = count_vect.fit_transform(X_train)
     X_test_counts = count_vect.transform(X_test)
```

Рисунок 5 – CountVectorizer

Проведём векторизацию признаков с помощью TfidfVectorizer (см. рисунок 6).

```
[17] tfidf_vect = TfidfVectorizer()
     X_train_tfidf = tfidf_vect.fit_transform(X_train)
     X_test_tfidf = tfidf_vect.transform(X_test)
```

Рисунок 6 – TfidfVectorizer

3.5 Обучение для CountVectorizer

Проведём обучение для CountVectorizer и LinearSVC (см. рисунок 7).

```
[18] gbc = LinearSVC()
     start_time = time.time()
     gbc.fit(X_train_counts, y_train)
     train_time = time.time() - start_time
     time_arr.append(train_time)
     pred_gbc_counts = gbc.predict(X_test_counts)
     print("Точность CountVectorizer и LinearSVC: ", accuracy_score(y_test, pred_gbc_counts))

Точность CountVectorizer и LinearSVC:  0.9883408071748879
```

Рисунок 7 – Обучение для CountVectorizer и LinearSVC

Проведём обучение для CountVectorizer и Logistic Regression (см. рисунок 8).

```
[19] lr = LogisticRegression(max_iter = 1000)
     start_time = time.time()
     lr.fit(X_train_counts, y_train)
     train_time = time.time() - start_time
     time_arr.append(train_time)
     pred_lr_counts = lr.predict(X_test_counts)
     print("Точность CountVectorizer и Logistic Regression: ", accuracy_score(y_test, pred_lr_counts))

Точность CountVectorizer и Logistic Regression:  0.9865470852017937
```

Рисунок 8 – Обучение для CountVectorizer и Logistic Regression

3.6 Обучение для TfidfVectorizer

Проведём обучение для TfidfVectorizer и LinearSVC (см. рисунок 9).

```
[23] gbc = LinearSVC()
     start_time = time.time()
     gbc.fit(X_train_tfidf, y_train)
     train_time = time.time() - start_time
     time_arr.append(train_time)
     pred_gbc_tfidf = gbc.predict(X_test_tfidf)
     print("Точность TfidfVectorizer и LinearSVC: ", accuracy_score(y_test, pred_gbc_tfidf))

     Точность TfidfVectorizer и LinearSVC:  0.9919282511210762
```

Рисунок 9 – Обучение для TfidfVectorizer и LinearSVC

Проведём обучение для TfidfVectorizer и Logistic Regression (см. рисунок 10).

```
[24] lr = LogisticRegression(max_iter = 1000)
     start_time = time.time()
     lr.fit(X_train_tfidf, y_train)
     train_time = time.time() - start_time
     time_arr.append(train_time)
     pred_lr_tfidf = lr.predict(X_test_tfidf)
     print("Точность TfidfVectorizer и Logistic Regression: ", accuracy_score(y_test, pred_lr_tfidf))

     Точность TfidfVectorizer и Logistic Regression:  0.9748878923766816
```

Рисунок 10 – Обучение для TfidfVectorizer и Logistic Regression

3.7 Анализ результатов

Выведем результаты (см. рисунок 11).

```
from tabulate import tabulate

data = [
    ["(CountVectorizer и LogisticRegression)", accuracy_score(y_test, pred_lr_counts), time_arr[0]],
    ["(CountVectorizer и LinearSVC)", accuracy_score(y_test, pred_gbc_counts), time_arr[1]],
    ["(TfidfVectorizer и LogisticRegression)", accuracy_score(y_test, pred_lr_tfidf), time_arr[2]],
    ["(TfidfVectorizer и LinearSVC)", accuracy_score(y_test, pred_gbc_tfidf), time_arr[3]]
]

sorted_data = sorted(data, key=lambda x: x[1], reverse=True)

print(tabulate(sorted_data,['Комбинация','Точность','Время обучения'], tablefmt="grid"))
```

```
+---------------------------------------+------------+------------------+
| Комбинация                            |   Точность |   Время обучения |
+=======================================+============+==================+
| (TfidfVectorizer и LinearSVC)         |   0.991928 |        0.0257075 |
+---------------------------------------+------------+------------------+
| (CountVectorizer и LinearSVC)         |   0.988341 |        0.115382  |
+---------------------------------------+------------+------------------+
| (CountVectorizer и LogisticRegression) |   0.986547 |        0.171523  |
+---------------------------------------+------------+------------------+
| (TfidfVectorizer и LogisticRegression) |   0.974888 |        0.0761209 |
+---------------------------------------+------------+------------------+
```

Рисунок 11 – Результаты

Представим результаты в виде таблицы (см. таблицу 1).

Таблица 1 – Результаты

Комбинация	**Точность, %**	**Время обучения, с**
TfidfVectorizer и LinearSVC	0.991928	0.0257075
CountVectorizer и LinearSVC	0.988341	0.1153820
CountVectorizer и LogisticRegression	0.986547	0.1715230
TfidfVectorizer и LogisticRegression	0.974888	0.0761209

Наивысшую точность можно наблюдать при комбинации TfidfVectorizer+LinearSVC (99,2%), наименьшую – при использовании комбинации TfidfVectorizer+LogisticRegression (97,5%). Наибольшее время обучения достигается при комбинации CountVectorizer+LogisticRegression (0,171 с), а наименьшее – при комбинации TfidfVectorizer+LinearSVC (0,026 с). Оба метода с использованием CountVectorizer выполняются дольше, чем аналогичные методы с TfidfVectorizer.