Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.03 Прикладная информатика**

ОТЧЕТ

по лабораторной работе № 4

Название:

Дисциплина: Прикладной анализ данных

Студент ИУ6-55Б _____ В.К. Полубояров
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ М.А. Кулаев
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

Цель работы: изучить различные методы очистки данных, методы предобработки текста.

1. Оставьте в выборках только строки с классами positive и negative.
2. Определите и реализуйте креативные методы очистки набора данных. Например, в твитах часто встречаются ссылки на аккаунты других пользователей, оформленные однотипным образом – кажется, что это лишняя информация.
3. Осуществите стемминг подготовленного набора данных и преобразуйте каждый твит в мешок слов. Помните, что кастомные преобразования обучаются только на train выборке. Если они необучаемые, то нужно взять один и тот же тип преобразования для обеих выборок (один и тот же метод из одной библиотеки).
4. Составьте Count-матрицу и рассчитайте на ней tf-idf. Обратите внимание, что tf-idf – это обучаемое преобразование, которое нужно зафитить на обучающих данных и применить затем к тестовым.
5. Обучите модели логистической регрессии и случайного леса на обучающей выборке, примените их к тестовым данным. Посчитайте качество на обучающих и тестовых данных, сравните результаты. Определите наиболее важные признаки (слова).
6. В пункте 3 вместо стемминга осуществите лемматизацию и проделайте пункты 3-5 с учетом другого типа подготовки данных.
7. Сравните результаты по качеству и по наиболее важным признакам (словам) между 2 обученными вариантами.

Задание 1. Оставьте в выборках только строки с классами positive и negative.

Определим два датафрейма (тестовый и тренировочный), взяв данные из источника, предоставленного в условии. Затем оставим только те данные, которые соответствуют классам positive и negative.

```
df_train = pd.read_csv('rusentitweet_train.csv')
df_test = pd.read_csv('rusentitweet_test.csv')

#оставляем только label = positive/negative
df_train = df_train[df_train['label'].isin(['positive', 'negative'])]
df_test = df_test[df_test['label'].isin(['positive', 'negative'])]
```

Рисунок 1 - Выделение необходимых классов в выборках

Задание 2. Очистка наборов данных.

В качестве очистки были использованы разнообразные методы, каждый из которых описан в комментариях на рисунке 2.

```
russian_stopwords = stopwords.words("russian")

def clean_tweet(tweet):
    tweet = re.sub(r'http\S+', '', tweet)   # Удаление ссылок
    tweet = re.sub(r'@\w+', '', tweet)      # Удаление упоминаний пользователей
    tweet = re.sub(r'#\w+', '', tweet)      # Удаление хештегов
    tweet = re.sub(r'\d+', '', tweet)       # Удаление чисел
    tweet = re.sub(r'[^\w\s]', '', tweet)   # Удаление пунктуации
    tweet = re.sub(r'[a-zA-Z]+', '', tweet) # Удаление слов на латинице
    tweet = re.sub(r'[^\w\s,]', '', tweet)  # Удаление эмодзи
    tweet = tweet.lower()                   # Приведение к нижнему регистру
    tweet = " ".join([word for word in tweet.split() if word not in russian_stopwords]) # Удаление стоп-слов
    tweet = re.sub(r'\s+', ' ', tweet).strip() # Удаление лишних пробелов
    tweet = re.sub(r'\b\w\b', '', tweet)   # Удаление слов из одной буквы
    return tweet

# Применение функции очистки к каждому твиту в наборах данных
df_train['text'] = df_train['text'].apply(clean_tweet)
df_test['text'] = df_test['text'].apply(clean_tweet)

# Применение функции очистки к каждому твиту в наборах данных
df_train['text'] = df_train['text'].apply(clean_tweet)
df_test['text'] = df_test['text'].apply(clean_tweet)
```

Рисунок 2 - Очистка наборов данных.

После изучения получившихся наборов данных было замечено, что некоторые твиты стали пустыми. Поэтому было принято решение удалить из выборки твиты, которые стали пустыми после очистки наборов данных.

```
# Удаление пустых твитов
df_train = df_train[df_train['text'].str.strip().astype(bool)]
df_test = df_test[df_test['text'].str.strip().astype(bool)]
```

Рисунок 3 - Удаление пустых твитов.

Задание 3. Стемминг подготовленного набора данных и преобразование в мешки слов.

В качестве стеммера был взят стеммер SnowballStemmer из библиотеки NLTK, алгоритм работы которого был разработан Мартином Портером.

(https://www.nltk.org/api/nltk.stem.SnowballStemmer.html?highlight=stopwords)

```
# Инициализируем стеммер для русского языка
stemmer = SnowballStemmer("russian")

def stem_tweet(tweet):
    # Разделяем твит на слова
    words = tweet.split()
    # Применяем стемминг к каждому слову
    stemmed_words = [stemmer.stem(word) for word in words]
    # Собираем обратно в строку
    return ' '.join(stemmed_words)

df_train_stem = df_train.copy()
df_test_stem = df_test.copy()

# Применяем функцию стемминга к каждому твиту
df_train_stem['text'] = df_train_stem['text'].apply(stem_tweet)
df_test_stem['text'] = df_test_stem['text'].apply(stem_tweet)
```

Рисунок 4 - Стемминг слов

После стемминга твитов было произведено преобразование в мешки слов.

```
# Инициализация CountVectorizer
vectorizer = CountVectorizer()

# Обучение на тренировочных данных и преобразование их в мешок слов
X_train = vectorizer.fit_transform(df_train_stem['text'])

# Преобразование тестовых данных в мешок слов
X_test = vectorizer.transform(df_test_stem['text'])
```

Рисунок 5 - Преобразование твитов в мешки слов

Задание 4. Расчет TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency): Это статистическая мера, используемая для оценки важности слова в контексте документа, который является частью коллекции документов. Основная идея TF-IDF заключается в том, чтобы уменьшить вес часто встречающихся слов во всём наборе документов, которые считаются менее информативными, и увеличить вес слов, которые редко встречаются, но могут быть ключевыми в определённых документах.

Term Frequency (TF): Это частота, с которой слово встречается в документе. Это просто количество раз, когда слово появляется в документе, поделённое на общее количество слов в этом документе.

Inverse Document Frequency (IDF): Это мера того, насколько информативным или редким является слово во всём наборе. Она уменьшает вес слов, которые часто встречаются в корпусе, и увеличивает вес редких слов.

Term Frequency (TF):

$$\mathrm{TF}(t,d) = \frac{\text{Количество вхождений слова } t \text{ в документе } d}{\text{Общее количество слов в документе } d}$$

Здесь t — это термин (слово), d — документ.

Inverse Document Frequency (IDF):

$$\mathrm{IDF}(t,D) = \log\left(\frac{\text{Общее количество документов в наборе } D}{\text{Количество документов, содержащих слово } t}\right)$$

D — это весь набор документов.

Рисунок 6 - Формула

Теперь рассчитаем матрицу TF-IDF для наших выборок. Сначала обучим ее на обучающей выборке и выполним преобразование для нее. Затем выполним преобразование на тестовой выборке.

```
# Создание объекта tf-idf transformer
tfidf_transformer = TfidfTransformer()

# Обучение на тренировочных данных и преобразование их
X_train_tfidf = tfidf_transformer.fit_transform(X_train)

# Преобразование тестовых данных
X_test_tfidf = tfidf_transformer.transform(X_test)
```

Рисунок 7 - TF-IDF

Задание 5. Логистическая регрессия и случайный лес.

Создадим соответствующие модели и выведем отчет по каждой из них.

```
# Создание модели логистической регрессии
logreg = LogisticRegression()
logreg.fit(X_train_tfidf, df_train['label'])

# Создание модели случайного леса
rf = RandomForestClassifier()
rf.fit(X_train_tfidf, df_train['label'])

# Предсказание на тестовых данных
logreg_pred = logreg.predict(X_test_tfidf)
rf_pred = rf.predict(X_test_tfidf)

# Оценка результатов
print("Логистическая регрессия:\n", classification_report(df_test['label'], logreg_pred))
print("Случайный лес:\n", classification_report(df_test['label'], rf_pred))
```

Рисунок 8 - Обучение моделей

Логистическая регрессия:

	precision	recall	f1-score	support
negative	0.73	0.93	0.82	656
positive	0.84	0.54	0.66	477
accuracy			0.76	1133
macro avg	0.79	0.73	0.74	1133
weighted avg	0.78	0.76	0.75	1133

Рисунок 9 - Отчет по логистической регрессии

Случайный лес:

	precision	recall	f1-score	support
negative	0.72	0.87	0.79	656
positive	0.75	0.54	0.63	477
accuracy			0.73	1133
macro avg	0.73	0.70	0.71	1133
weighted avg	0.73	0.73	0.72	1133

Рисунок 10 - Отчет по случайному лесу

Исходя из результатов, обе модели показали схожие значения точности (precision) и F1-меры. Важно заметить, что хотя общая точность (accuracy) у обеих моделей одинакова (0.73), существуют значимые различия в точности и полноте (recall) для классов 'positive' и 'negative':

- Логистическая регрессия:
    - Имеет более высокую точность для положительного класса (0.84 против 0.75).
    - Для отрицательного класса полнота выше (0.93 против 0.87), но точность немного ниже (0.73 против 0.72).
- Случайный лес:

- Имеет более низкую точность для положительного класса и ту же полноту, что и логистическая регрессия.
- Для отрицательного класса точность чуть ниже и полнота меньше, чем у логистической регрессии.

Если судить по F1-мере, которая является гармоническим средним между точностью и полнотой, логистическая регрессия показывает лучшие результаты для положительного класса (0.66 против 0.63), в то время как случайный лес показывает лучший результат для отрицательного класса (0.79 против 0.82).

В целом, можно утверждать, что:

- Логистическая регрессия лучше справляется с определением положительных твитов, что может быть предпочтительнее, если важнее точно идентифицировать положительные твиты.
- Случайный лес показывает себя чуть лучше в определении отрицательных твитов, что может быть важнее в ситуациях, когда необходимо выявить отрицательные настроения.

Теперь определим 20 самых важных слов для каждой из моделей. Стоит заметить, что для модели логистической регрессии были взяты 10 слов с максимальным положительным весом и 10 слов с максимально отрицательном весом, что соответствует позитивным и отрицательным классам твитов.

```python
# Для логистической регрессии
feature_names = vectorizer.get_feature_names_out()
logreg_coef = logreg.coef_[0]
sorted_features = sorted(zip(logreg_coef, feature_names), reverse=True)
print("Наиболее важные положительные слова для логистической регрессии:")
sorted_words = [word for coef, word in sorted_features]
sorted_words_string = ' '.join(sorted_words[:10])
print(sorted_words_string)

sorted_features = sorted(zip(logreg_coef, feature_names))
print("Наиболее важные отрицательные слова для логистической регрессии:")
sorted_words = [word for coef, word in sorted_features]
sorted_words_string = ' '.join(sorted_words[:10])
print(sorted_words_string)

# Для случайного леса
rf_importances = rf.feature_importances_
# Получаем индексы наиболее важных признаков в порядке убывания их важности
sorted_indices = np.argsort(rf_importances)[::-1]
# Извлекаем соответствующие слова
top_words = [feature_names[i] for i in sorted_indices[:20]]
# Преобразуем список слов в строку, разделяя слова пробелами
top_words_string = ' '.join(top_words)
print("Наиболее важные слова для случайного леса:")
print(top_words_string)
```

Рисунок 11 - Код определения самых важных признаков (слов)

Результаты представлены на рисунке 12.

```
Наиболее важные положительные слова для логистической регрессии:
любл красив прекрасн лучш мил классн крут нрав ва рад
Наиболее важные отрицательные слова для логистической регрессии:
блят пиздец нах сук хуйн вообщ заеба ненавиж грустн ужасн
Наиболее важные слова для случайного леса:
любл красив блят прекрасн крут лучш мил эт классн пиздец нрав хорош рад ва хоч сук очен любим обожа нах
```

Рисунок 12 - Наиболее важные признаки для каждой из моделей

Задание 6. Лемматизация

Для лемматизации воспользуемся Mystem, разработанным сотрудниками Яндекса для поискового сервиса. (https://github.com/nlpub/pymystem3)

```python
# Инициализируем Mystem
mystem = Mystem()

def lemmatize_text(text):
    # Применяем лемматизацию к тексту и объединяем обратно в строку
    lemmas = mystem.lemmatize(text)
    return ''.join(lemmas).strip()

df_train_lem = df_train.copy()
df_test_lem = df_test.copy()

# Применяем функцию лемматизации к столбцу с текстом в DataFrame
df_train_lem['text'] = df_train_lem['text'].apply(lemmatize_text)
df_test_lem['text'] = df_test_lem['text'].apply(lemmatize_text)
```

Рисунок 13 - Лемматизация

Остальной код будет идентичным, так как он повторяет пункты 3-5.

Логистическая регрессия:

	precision	recall	f1-score	support
negative	0.74	0.92	0.82	656
positive	0.84	0.56	0.67	477
accuracy			0.77	1133
macro avg	0.79	0.74	0.75	1133
weighted avg	0.79	0.77	0.76	1133

Рисунок 14 - Логистическая регрессия для лемматизации

Случайный лес:

	precision	recall	f1-score	support
negative	0.73	0.87	0.79	656
positive	0.76	0.55	0.63	477
accuracy			0.74	1133
macro avg	0.74	0.71	0.71	1133
weighted avg	0.74	0.74	0.73	1133

Рисунок 15 - Случайный лес для лемматизации

```
Наиболее важные положительные слова для логистической регрессии:
любить хороший красивый милый прекрасный классный вау нравиться крутой любовь
Наиболее важные отрицательные слова для логистической регрессии:
блять пиздец нахуй сука умирать вообще хуйня ненавидеть уходить сдыхать
Наиболее важные слова для случайного леса:
любить хороший блять красивый это пиздец нравиться вау прекрасный хотеть любовь милый классный крутой самый очень
вообще красиво умирать рад
```

Рисунок 15 - Самые важные слова при лемматизации

Задание 7. Сравнение полученных результатов.

Исходя из полученных результатов, можно сказать, что модель логистической регрессии стала незначительна точнее, но разница несущественная.

Случайный лес тоже отличается по показателем незначительно ±0.01 на различных показателях.

При этом все 4 модели крайне близки друг к другу в плане метрик, что говорит о том, что для данных входных данных можно было бы использовать любую из четырех.

Важные слова в основной массе идентичны как при стемминге, так и при лемматизации.

Однако, важно заметить, что стемминг работает в несколько раз быстрее лемматизации, что может быть существенно при бОльшем количестве входных данных.

Главный плюс лемматизации над стеммингом заключается в том, что итоговые слова являются осмысленными: можно понять часть речи, вид (в случае глагола) и так далее. В то время как при стемминге обрезается часть слова и понять исходный смысл слова или часть его речи бывает невозможно.

Вывод

В ходе выполнения лабораторной работы были изучены различные методы очистки текста и два основных метода предобработки текста: стемминг и лемматизация.