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

Дисциплина: Прикладной анализ данных

Название: Определение семантической окраски твитов

Студент ИУ6-55Б _____ А.Д. Шевченко
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2023

Цель: подготовка и анализ набора данных для обучения моделей классификации текста с использованием различных методов предобработки.

Формулировка:

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.

Основная часть

1. По ссылке, данной в задании, были скачены 2 csv-файла: один с тренировочными данными, другой – с тестовыми. Далее в датафрейме были оставлены только те строки, у который label равняется либо positive, либо negative.

```python
import pandas as pd

# Загружены тренировочные и тестовые данные
data_train = pd.read_csv('rusentitweet_train.csv')
data_test = pd.read_csv('rusentitweet_test.csv')

# Оставлены только строки с label 'positive' или 'negative'
data_train = data_train[data_train['label'].isin(['positive', 'negative'])]
data_test = data_test[data_test['label'].isin(['positive', 'negative'])]

data_train.head()
```

	text	label	id
0	Помойму я вкрашилась в Чимина🤧 https://t.co/t2...	positive	1282311169534038016
5	@buybread_ я не с порядке!!!!	negative	1335130757044563971
10	@ange1flyhigh В следующий раз буду до победног...	positive	1215370396465291267
15	@LimitaVIP Удивительный гімн...\r\nУдивительно...	negative	1253799540848762887
17	я срала на эту биологию	negative	1339418979887173632

Рисунок 1 – загрузка данных

2. Следующим шагом была произведена очистка данных от веб-ссылок, ссылок на твиттер-аккаунты и хэштегов (рисунок 2).

```python
import re

# Функция для очистки текста от веб-ссылок, ссылок на твиттер-аккаунты и хэштегов
def clean_data(text):
  text = re.sub(r'(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)', '', text) # Убираем веб-ссылки
  text = re.sub(r'@[^\s]+', '', text) # Убираем ссылки на твиттер-аккаунты
  text = re.sub(r'#\w+', '', text) # Убираем хэштеги
  return text

# Применение функции для очистки к столбцу text тренировочного и тестового датафреймов
data_train['text'] = data_train['text'].apply(clean_data)
data_test['text'] = data_test['text'].apply(clean_data)

data_train.head()
```

	text	label	id
0	Помойму я вкрашилась в Чимина🤧	positive	1282311169534038016
5	я не с порядке!!!!	negative	1335130757044563971
10	В следующий раз буду до победного ее закрыват...	positive	1215370396465291267
15	Удивительный гімн...\r\nУдивительно, что пока...	negative	1253799540848762887
17	я срала на эту биологию	negative	1339418979887173632

Рисунок 2 – очистка данных

3. Следующим шагом был осуществлен стемминг всех твитов с помощью библиотеки nltk, то есть каждое слово было приведено в свою основу (рисунок 3).

```
from nltk.stem.snowball import SnowballStemmer

# Инициализация объекта-стеммера
stemmer = SnowballStemmer('russian')

# Функция для стемминга
def stemming(text):
  return ' '.join([stemmer.stem(s) for s in text.split()])

# Применение стемминга
stemmed_data_train = data_train.copy()
stemmed_data_test = data_test.copy()
stemmed_data_train['text'] = stemmed_data_train['text'].apply(stemming)
stemmed_data_test['text'] = stemmed_data_test['text'].apply(stemming)

stemmed_data_train.head()
```

	text	label	id
0	помойм я вкраш в чимина🤧	positive	1282311169534038016
5	я не с порядке!!!!	negative	1335130757044563971
10	в след раз буд до победн е закрыва пожела удачи!	positive	1215370396465291267
15	удивительн гимн... удивительно, что пок ещ не ...	negative	1253799540848762887
17	я срал на эт биолог	negative	1339418979887173632

Рисунок 3 – стемминг твитов

Далее с помощью библиотеки sklearn наши твиты были преобразованы в мешок слов, а также была получена count-матрица (рисунок 4). Мешок слов – упрощенное представление текста, которое показывает, какие слова встретились в тексте, но при этом не учитывает их порядок.

```python
from sklearn.feature_extraction.text import CountVectorizer

# Инициализация объекта для преобразования в мешок
vectorizer = CountVectorizer()

# Осуществление преобразования
data_train_count_M = vectorizer.fit_transform(stemmed_data_train['text'])
data_test_count_M = vectorizer.transform(stemmed_data_test['text'])

print(data_train_count_M)
```

```
  (0, 6142)     1
  (0, 1083)     1
  (0, 9301)     1
  (1, 4534)     1
  (1, 6251)     1
  (2, 7587)     1
  (2, 6923)     1
  (2, 811)      1
  (2, 2009)     1
  (2, 5799)     1
  (2, 2583)     1
  (2, 5986)     1
  (2, 8621)     1
  (3, 4534)     1
  (3, 8627)     1
```

Рисунок 4 – преобразование в мешок слов

4. Следующим этапом стало вычисление tf-idf. TF (Term Frequency) – отношение количества вхождений слова в документ к общему количеству слов в документе. IDF (Inverse Document Frequency) – логарифм отношения общего количества документов к количеству документов, в которые выходит рассматриваемое слово.

```
from sklearn.feature_extraction.text import TfidfTransformer

# Инициализация объекта для расчета tf-idf
transformer = TfidfTransformer()

# Осуществление расчета tf-idf
data_train_tfidf = transformer.fit_transform(data_train_count_M)
data_test_tfidf = transformer.transform(data_test_count_M)

print(data_train_tfidf)
```

```
  (0, 9301)     0.5862800063736923
  (0, 6142)     0.5862800063736923
  (0, 1083)     0.5590630628586786
  (1, 6251)     0.9479517211905822
  (1, 4534)     0.318414092483063
  (2, 8621)     0.4230979525398923
  (2, 7587)     0.35023749583944885
  (2, 6923)     0.25018279336641547
  (2, 5986)     0.40345642809695176
  (2, 5799)     0.4230979525398923
  (2, 2583)     0.40345642809695176
  (2, 2009)     0.2529918097669453
  (2, 811)      0.25915145565853115
  (3, 9337)     0.15455285008668587
  (3, 8676)     0.43704293918051396
  (3, 8628)     0.40235884554866047
  (3, 8627)     0.39119306136286297
```

Рисунок 7 – матрица tf-idf

TF-IDF матрица отображает следующую информацию слева направо: номер документа, id слова и вес tf-idf, который показывает, насколько важно слово, т.е. как часто оно вообще встречается в наших документах.

5. На следующем шаге были сделаны создание, обучение и проверка работы моделей логистической регрессии и случайного леса. Также варианты таргетов «positive» и «negative» были заменены на 1 и 0 соответственно. Работа моделей была проверена как на тренировочных, так и на тестовых данных.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Превращение слов positive и negative в 1 и 0
label_convert = {'positive': 1, 'negative': 0}
data_train_targets = data_train['label'].map(label_convert)
data_test_targets = data_test['label'].map(label_convert)

# Создание моделей логистической регрессии и случайного леса
logistic_reg = LogisticRegression()
random_forest = RandomForestClassifier()

# Обучение моделей
logistic_reg.fit(data_train_tfidf, data_train_targets)
random_forest.fit(data_train_tfidf, data_train_targets)

# Проверка на тестовых данных
logistic_reg_predict = logistic_reg.predict(data_test_tfidf)
random_forest_predict = random_forest.predict(data_test_tfidf)

# Проверка на тренировочных данных
logistic_reg_predict_train = logistic_reg.predict(data_train_tfidf)
random_forest_predict_train = random_forest.predict(data_train_tfidf)
```

Рисунок 8 – код создания, обучения и использования для предсказаний моделей случайного леса и логистической регрессии

На рисунке 9 изображен отчет по работе моделей на тестовых данных, созданный с помощью библиотеки sklearn.

```python
from sklearn.metrics import classification_report

# Отчет по работе модели логистической регрессии на тестовых данных
print('\tЛогистическая регрессия\n\n', classification_report(data_test_targets, logistic_reg_predict))

# Отчет по работе модели случайного леса на тестовых данных
print('\tСлучайный лес\n\n', classification_report(data_test_targets, random_forest_predict))
```

```
        Логистическая регрессия

              precision    recall  f1-score   support

           0       0.73      0.91      0.81       660
           1       0.82      0.55      0.66       483

    accuracy                           0.76      1143
   macro avg       0.78      0.73      0.74      1143
weighted avg       0.77      0.76      0.75      1143

        Случайный лес

              precision    recall  f1-score   support

           0       0.72      0.89      0.79       660
           1       0.77      0.52      0.62       483

    accuracy                           0.73      1143
   macro avg       0.74      0.70      0.71      1143
weighted avg       0.74      0.73      0.72      1143
```

Рисунок 9 – отчет по работе моделей на тестовых данных

На рисунке 10 изображен отчет по работе моделей на тренировочных данных, так же сгенерированный с помощью библиотеки sklearn.

```
from sklearn.metrics import classification_report

# Отчет по работе модели логистической регрессии на тренировочных данных
print('\tЛогистическая регрессия\n\n', classification_report(data_train_targets, logistic_reg_predict_train))

# Отчет по работе модели случайного леса на тренировочных данных
print('\tСлучайный лес\n\n', classification_report(data_train_targets, random_forest_predict_train))
```

```
        Логистическая регрессия

              precision    recall  f1-score   support

           0       0.88      0.98      0.92      2638
           1       0.96      0.81      0.88      1931

    accuracy                           0.91      4569
   macro avg       0.92      0.90      0.90      4569
weighted avg       0.91      0.91      0.91      4569

        Случайный лес

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2638
           1       1.00      1.00      1.00      1931

    accuracy                           1.00      4569
   macro avg       1.00      1.00      1.00      4569
weighted avg       1.00      1.00      1.00      4569
```

Рисунок 10 – отчет по работе моделей на тренировочных данных

Рассмотрим результаты работы обеих моделей сначала на тестовых данных. Изучая отчет, можно увидеть следующее:

- Для модели логистической регрессии:
  - F1-мера, являющаяся гармоническим средним между точностью и полнотой, и особенно полнота значительно выше для отрицательного класса, чем для положительного. Это может свидетельствовать о том, что модель лучше справляется с выявлением негативных твитов. Однако точность для положительного класса чуть повыше, чем для отрицательного.
- Для модели случайного леса:
  - Результаты очень схожи с моделью логистической регрессии, так что вывод можно сделать аналогичный: модель случайного леса лучше справляется с выявлением негативных твитов, нежели с выявлением позитивным.

Также, в целом, можно сказать, что обе модели вполне неплохо себя

продемонстрировали в решении нашей задачи.

Далее рассмотрим результаты работы на тренировочных данных. Тут ситуация поинтереснее:

- Для модели логистической регрессии:
  - Модель показывает хорошие результаты, однако полнота для отрицательного класса значительно выше, чем для положительного, а точность чуть пониже. В целом, все параметры, в частности F1-мера, находятся на вполне высоком уровне, так что можем сказать, что модель хорошо справилась с задачей.
- Для модели случайного леса:
  - Модель просто идеально справилась с задачей, поскольку все три ее параметра равны единице (точность, полнота, F1-мера).

При сравнении результатов работы моделей на тестовых и на тренировочных данных становится очевидно, что переобучение присутствует у обеих моделей, особенно у случайного леса.

Следующим этапом стало определение 20 наиболее важных слов для каждой модели (рисунок 11).

```
# Получение имен признаков
feature_names = vectorizer.get_feature_names_out()

# Получение и сортировка абсолютных значений коэффициентов модели лог. регрессии
lr_coefs = logistic_reg.coef_[0]
lr_coefs = abs(lr_coefs).argsort()[::-1]

# 20 самых важных слов для модели лог. регрессии
lr_most_important_features = [feature_names[i] for i in lr_coefs[:20]]
print('20 самых важных слов для модели лог. регрессии:\n', lr_most_important_features)

print('')

# Получение и сортировка коэффициентов модели случайного леса
rf_feature_importances = random_forest.feature_importances_
rf_feature_importances = rf_feature_importances.argsort()[::-1]

# 20 самых важных слов для модели случайного леса
rf_most_important_features = [feature_names[i] for i in rf_feature_importances[:20]]
print('20 самых важных слов для модели случайного леса:\n', rf_most_important_features)
```

```
20 самых важных слов для модели лог. регрессии:
 ['не', 'любл', 'хорош', 'пиздец', 'красив', 'блят', 'нет', 'нрав', 'мил', 'классн', 'крут', 'нах', 'рад', 'хорошо', 'сук', 'прекрасн', 'вау', 'ненавиж',

20 самых важных слов для модели случайного леса:
 ['не', 'так', 'любл', 'эт', 'хорош', 'что', 'нет', 'пиздец', 'все', 'красив', 'как', 'блят', 'мо', 'на', 'ты', 'теб', 'за', 'он', 'мен', 'нрав']
```

Рисунок 11 – 20 самых важных слов по мнению обеих моделей

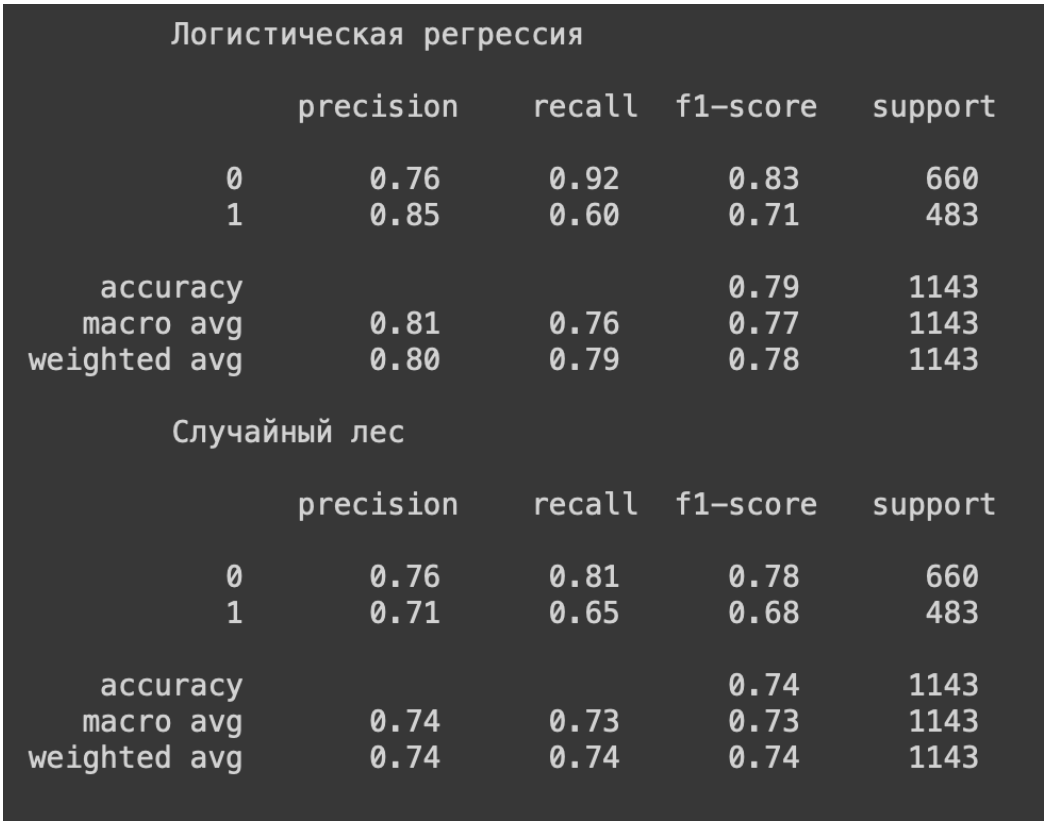
6. Далее стемминг, реализованный в пункте 3, был изменен на лемматизацию. Для лемматизации использовалась библиотека pymystem.

```
from pymystem3 import Mystem

# Инициализация объекта для лемматизации
mystem = Mystem()

# Функция лемматизации
def lemmatization(text):
  lemmas = mystem.lemmatize(text)
  res = [l for l in lemmas if l.isalnum()]
  return ' '.join(res)

# Применение лемматизации
lemmed_data_train = data_train.copy()
lemmed_data_test = data_test.copy()
lemmed_data_train['text'] = lemmed_data_train['text'].apply(lemmatization)
lemmed_data_test['text'] = lemmed_data_test['text'].apply(lemmatization)

lemmed_data_train.head()
```

	text	label	id
0	помойма я вкрашиться в чимин	positive	1282311169534038016
5	я не с порядок	negative	1335130757044563971
10	в следующий раз быть до победный она закрывать...	positive	1215370396465291267
15	удивительный гімн удивительно что пока еще не ...	negative	1253799540848762887
17	я срать на этот биология	negative	1339418979887173632

Рисунок 12 – лемматизация данных

Далее снова были рассмотрены результаты работы моделей на тренировочных и на тестовых данных, но уже с примененной до этого лемматизацией вместо стемминга. Результаты на тестовых и тренировочных данных данных изображены на рисунках 13-14.



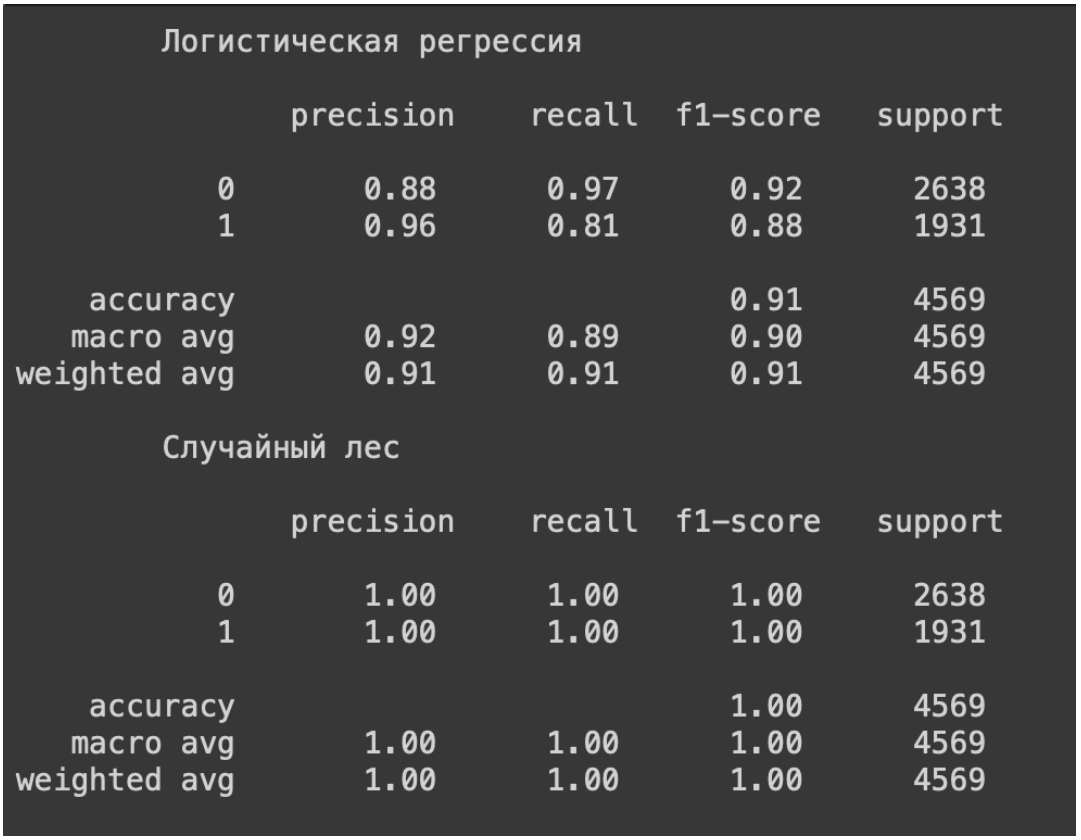
Логистическая регрессия

	precision	recall	f1-score	support
0	0.76	0.92	0.83	660
1	0.85	0.60	0.71	483
accuracy			0.79	1143
macro avg	0.81	0.76	0.77	1143
weighted avg	0.80	0.79	0.78	1143

Случайный лес

	precision	recall	f1-score	support
0	0.76	0.81	0.78	660
1	0.71	0.65	0.68	483
accuracy			0.74	1143
macro avg	0.74	0.73	0.73	1143
weighted avg	0.74	0.74	0.74	1143

Рисунок 13 – результат работы моделей на тестовых данных с лемматизацией



Логистическая регрессия

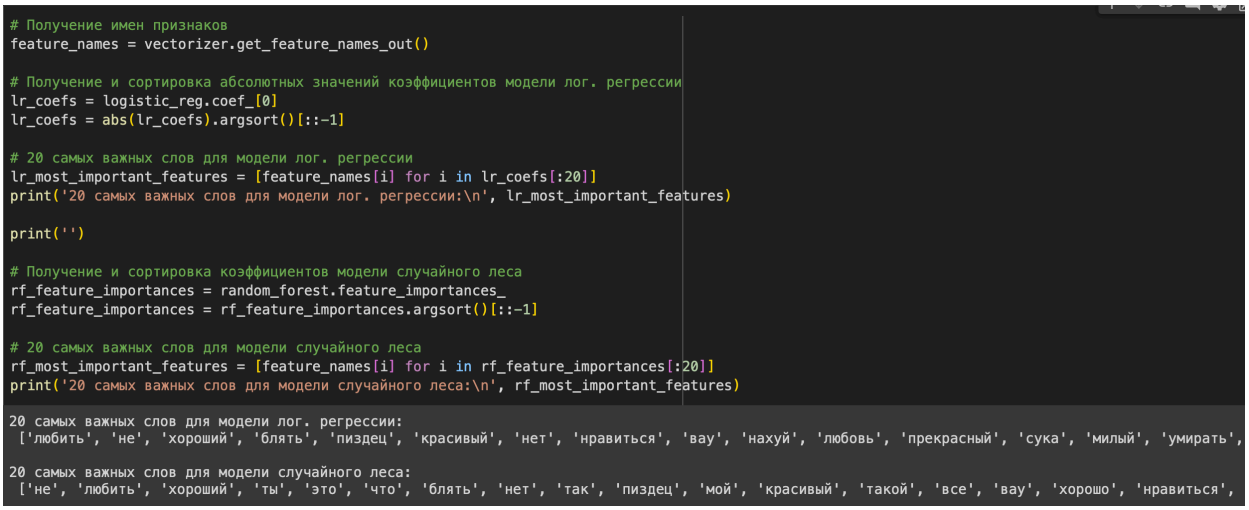
	precision	recall	f1-score	support
0	0.88	0.97	0.92	2638
1	0.96	0.81	0.88	1931
accuracy			0.91	4569
macro avg	0.92	0.89	0.90	4569
weighted avg	0.91	0.91	0.91	4569

Случайный лес

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2638
1	1.00	1.00	1.00	1931
accuracy			1.00	4569
macro avg	1.00	1.00	1.00	4569
weighted avg	1.00	1.00	1.00	4569

Рисунок 14 – результат работы моделей на тренировочных данных с лемматизацией

Рассмотрим 20 самых важных слов по мнению обеих моделей с примененной лемматизацией (рисунок 15).



```
# Получение имен признаков
feature_names = vectorizer.get_feature_names_out()

# Получение и сортировка абсолютных значений коэффициентов модели лог. регрессии
lr_coefs = logistic_reg.coef_[0]
lr_coefs = abs(lr_coefs).argsort()[::-1]

# 20 самых важных слов для модели лог. регрессии
lr_most_important_features = [feature_names[i] for i in lr_coefs[:20]]
print('20 самых важных слов для модели лог. регрессии:\n', lr_most_important_features)

print('')

# Получение и сортировка коэффициентов модели случайного леса
rf_feature_importances = random_forest.feature_importances_
rf_feature_importances = rf_feature_importances.argsort()[::-1]

# 20 самых важных слов для модели случайного леса
rf_most_important_features = [feature_names[i] for i in rf_feature_importances[:20]]
print('20 самых важных слов для модели случайного леса:\n', rf_most_important_features)
```

```
20 самых важных слов для модели лог. регрессии:
 ['любить', 'не', 'хороший', 'блять', 'пиздец', 'красивый', 'нет', 'нравиться', 'вау', 'нахуй', 'любовь', 'прекрасный', 'сука', 'милый', 'умирать',

20 самых важных слов для модели случайного леса:
 ['не', 'любить', 'хороший', 'ты', 'это', 'что', 'блять', 'нет', 'так', 'пиздец', 'мой', 'красивый', 'такой', 'все', 'вау', 'хорошо', 'нравиться',
```

Рисунок 15 – 20 самых важных слов по мнению каждой модели с лемматизацией

Сравнивая отчеты по работе моделей с примененными к тексту стеммингом и лемматизацией, можно увидеть, что разницы между результатами практически нет. Это говорит о том, что, в частности, в нашем случае оба способа предобработки текста одинаково хороши. Однако, в контексте понимания программистом результатов работы моделей, можно отдать предпочтение в пользу лемматизации, поскольку при выборе такого способа предобработанный текст гораздо более понятный и наглядный, чем при выборе стемминга.

Вывод: в ходе выполнения лабораторной работы были изучены основные методы предобработки данных – стемминг и лемматизация – и получено понимание работы с текстом на примере решения задачи о семантической окраски твитов.