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 **09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА**

О т ч е т

по лабораторной работе № 4

Вариант 3

Название: Автоматическая обработка текстов

Дисциплина: Прикладной анализ данных

Студент гр. ИУ6-51Б _____ К. А. Татаренко
(Подпись, дата) (И.О. Фамилия)

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

Москва, 2023

Цель работы – изучение методов предобработки текстовых документов (очистки), применение лемматизации и стемминга к тексту, построение модели на основе tf-idf, сравнение полученных результатов.

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

```
df2 = pd.DataFrame(columns=['text', 'label'])
df1 = pd.DataFrame(columns=['text', 'label'])
df_lem_train = pd.DataFrame(columns=['text', 'label'])
df_lem_test = pd.DataFrame(columns=['text', 'label'])

cou = 0

# достаем обучающую выборку и загружаем в 2 ДатаФрейма: для леммати
df_train = pd.read_csv('/Users/mac/Downloads/rusentitweet_train.csv
for i, row in df_train.iterrows():
    if row[1] in ("negative", "positive"):
        if row[1]=='negative':
            cou+=1
        df2.loc[len(df2.index)] = row
        df_lem_train.loc[len(df_lem_train.index)] = row

df_train = df2
print('Процент negative в train: ', cou/df_train.shape[0])

cou = 0
# достаем тестову выборку и загружаем в 2 ДатаФрейма: для лемматиза
df_test = pd.read_csv('/Users/mac/Downloads/rusentitweet_test.csv')
for i, row in df_test.iterrows():
    if row[1] in ("negative", "positive"):
        df1.loc[len(df1.index)] = row
        if row[1]=='negative':
            cou+=1
        df_lem_test.loc[len(df_lem_test.index)] = row
df_test = df1
print('Процент negative в test: ', cou/df_test.shape[0])


print(df_train)
```

```
Процент negative в train:  0.5773692274020573
Процент negative в test:  0.5774278215223098
```

Рисунок 1 – Код получения определенных строк и определения процентного соотношения

Было оставлены строки, которые помечены positive или negative. 4 набора (2 тестовых и 2 обучающи) из-за того, что если сделать один и второй равный этому, то второй сохрнаится как ссылка на 1 и все изменения применятся и ко второму, поэтому было создано 2 обучающих – для задачи лемматизации и стемминга (аналогично и для тестовых).

Можем заметить, что процент positive и negative строк примерно одинаковое (примерно сбалансированное) – 42 на 58 %. Это значит, что мы **можем применять метрику accuracy** для оценки качества модели.

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.**

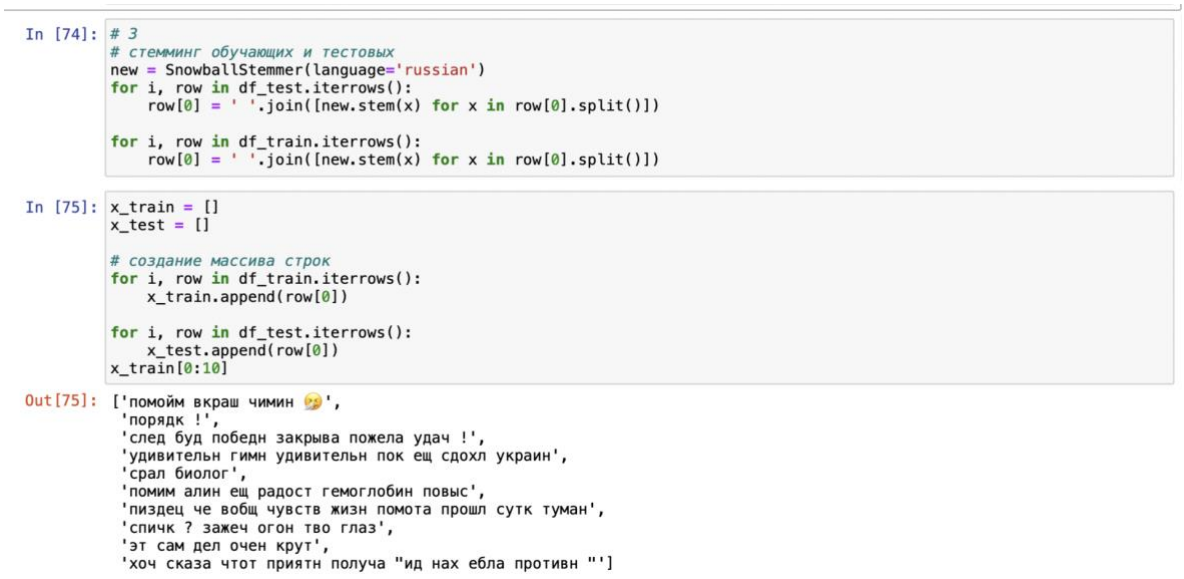
Была составлена функция очистки данных, в которой осуществлено:

- Приведение к нижнему регистру
- Обработка твита (удаление ссылок, упоминаний других пользователей, хэштегов(так как после # словапишутсяподряд)
- Удаление знаков препинания (кроме ! и ? знаков, так как они могут влиять на окрас эмоциональный окрас твита)
- Удаление стоп-слов (полученных из библиотеки)
- Удаление цифр (например, дат), так как они не вносят никакой эмоциональной состовляющей
- Удаление подряд идущих символов, не считая согласных (согласные не берем, так как в русском языке есть слова с 2 подряд согласными, удаляем подобное – яяяя->я, тыыыы->ты, !!!!->!, 😖 😖 😖-> 😖)
- Отделение смайликов от слов (было замечено, что люди пишут смайлики сразу после слов, не отделяя их пробелами: чимина😖 -> чимина 😖)

```
# 2 Функция очистки
def my_clear(df):
    s=[]
    s1 = ""
    regs = [r'https?:\/\/.\S+', r'#\S+', r'@\S+', r'\\r\\n', r'[-.)(,:]']
    glasn = ['о', 'а', 'я', 'а', 'и', 'е', 'ы']
    signs = ['!', '_', '.']

    # приведение всего к нижнему регистру
    for i, row in df.iterrows():
        row[0] = row[0].lower()


    for i, row in df.iterrows():
        # удаление ссыло, хэштегов(и знака решетки и последующего хэштега, потому что слова после хэштэга пишутся сл
        # удаление отмеченных пользователей(@name), удаление знаков пунктуации (кроме ! ?)
        for reg in regs:
            row[0] = re.sub(reg, "", row[0])

        # удаление стоп-слов
        row[0] = ' '.join([x for x in row[0].split() if x not in stop])

        # удаление повторящихся подряд смайликов и знаков препинания (оставшихся)
        # удаление подряд идущих гласных (ыыыыы, яяяяяя, ааааааа и тд)
        # удаление цифр, так как по сути цифры не имеют никакого значения в данном анализе
        r = row[0]
        for t in range(0, len(row[0])):
            if not(not(r[t].isalpha()) and r[t] == r[t-1]) and not((r[t] in glasn) and r[t] == r[t-1]) and not(r[t].
                s1+=r[t]
        row[0] = s1

        s1 = ""

        r = row[0] + "  "

        #отделение смайлика от слова (чимина🥺 -> чимина 🥺)
        for t in range(0, len(row[0])):
            if not(r[t+1].isalpha()) and r[t]!= " " and not(r[t+1].isdigit()) and (r[t+2].isalpha() or r[t+2]==" "):
                s1+=r[t] + " "
            else:
                s1+=r[t]
        row[0] = s1

        s = []
        s1=""
    return df
```

Рисунок 2 – Самописная функция предобработки текста

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).**



```
In [74]:
# 3
# стемминг обучающих и тестовых
new = SnowballStemmer(language='russian')
for i, row in df_test.iterrows():
    row[0] = ' '.join([new.stem(x) for x in row[0].split()])

for i, row in df_train.iterrows():
    row[0] = ' '.join([new.stem(x) for x in row[0].split()])

In [75]:
x_train = []
x_test = []

# создание массива строк
for i, row in df_train.iterrows():
    x_train.append(row[0])

for i, row in df_test.iterrows():
    x_test.append(row[0])
x_train[0:10]

Out[75]:
['помойм вкраш чимин 🥺',
 'порядк !',
 'след буд победн закрыва пожела удач !',
 'удивительн гимн удивительн пок ещ сдохл украин',
 'срал биолог',
 'помим алин ещ радост гемоглобин повыс',
 'пиздец че вобщ чувств жизн помота прошл сутк туман',
 'спичк ? зажеч огон тво глаз',
 'эт сам дел очен крут',
 'хоч сказа чтот приятн получа "ид нах ебла противн "']
```

Рисунок 3 – Стемминг текста, сбор всех строк в массив

Стемминг данных осуществляется за счет применения медота stem из библиотеки SnowBallStemmer. Далее все записи записываются в один массив, где каждый элемент – отдельный твит.

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.**

Получим tf-idf с помощью библиотеки sklearn, а именно с помощью объекта TfidVectorizer. TfidfVectorizer преобразует коллекцию необработанных документов в матрицу функций TF-IDF.

```
In [93]: # 4
vec = TfidfVectorizer(stop_words=None)

# tf-idf test and train
matr = vec.fit_transform(x_train)
matr_test = vec.transform(x_test)
print(matr[0:3])
```

```
  (0, 7572)     0.5346549574878128
  (0, 823)      0.5831906314872191
  (0, 4989)     0.6115821807240984
  (1, 5072)     1.0
  (2, 7006)     0.39675554325904605
  (2, 4857)     0.4497167039101802
  (2, 2037)     0.43418293347176135
  (2, 4702)     0.47161032368447664
  (2, 604)      0.2853237257897314
  (2, 6170)     0.3847066148312607
```

Рисунок 4 – Получение tf-idf

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).**

При первоначальном решении (при значениях параметров по умолчанию) модель была сильна переобучена, что видно на рисунке

```
In [69]: # 5 logreg
model_logreg = LogisticRegression()
model_logreg.fit(matr, df_train['label'])
y_logreg = model_logreg.predict(matr_test)

accuracy = accuracy_score(df_test['label'], y_logreg)
print(accuracy)
```

```
0.7515310586176728
```

```
In [70]: model_logreg = LogisticRegression()
model_logreg.fit(matr, df_train['label'])
y_logreg = model_logreg.predict(matr)

accuracy = accuracy_score(df_train['label'], y_logreg)
print(accuracy)
```

```
0.9520682862770847
```

Рисунок 5 – Переобученная модель

По этой причине был произведен перебор параметра регуляризации (для логистической регрессии) и максимальной глубины дерева (для случайного леса) для того, чтобы точность моделей на тестовых и обучающих данных была примерно равна.

```
# 5 logreg
model_logreg = LogisticRegression(C=0.31)
model_logreg.fit(matr, df_train['label'])
y_logreg = model_logreg.predict(matr_test)

accuracy = accuracy_score(df_test['label'], y_logreg)
print(accuracy)
```

```
0.7147856517935258
```

```
model_logreg = LogisticRegression(C=.31)
model_logreg.fit(matr, df_train['label'])
y_logreg = model_logreg.predict(matr)

accuracy = accuracy_score(df_train['label'], y_logreg)
print(accuracy)
```

```
0.7089078572991901
```

```
# 5 top logreg
importance = model_logreg.coef_
abs_importance = abs(importance)
sorted_index = abs_importance.argsort()
sorted_index = sorted_index[0][::-1]
# топ-10 значимых слов (горячая десятка)
np.array(vec.get_feature_names_out())[sorted_index[0:10]]
```

```
array(['любл', 'красив', 'блят', 'пиздец', 'мил', 'крут', 'ва',
       'прекрасн', 'лучш', 'хорош'], dtype=object)
```

Рисунок 6 – Полученные показатели модели после подбора параметра регуляризации логистической регрессии

```
# 5 random_forest
model_random_forest = RandomForestClassifier(max_depth=35)
model_random_forest.fit(matr, df_train['label'])
y_forest = model_random_forest.predict(matr)

accuracy = accuracy_score(df_train['label'], y_forest)
print(accuracy)
```

```
0.6596629459400306
```

```
model_random_forest = RandomForestClassifier(max_depth=35)
model_random_forest.fit(matr, df_train['label'])
y_forest = model_random_forest.predict(matr_test)

accuracy = accuracy_score(df_test['label'], y_forest)
print(accuracy)
```

```
0.6622922134733158
```

```
# 5 top random_forest
importance = model_random_forest.feature_importances_
abs_importance = abs(importance)
sorted_index = abs_importance.argsort()[::-1]
# топ-10 значимых слов
np.array(vec.get_feature_names_out())[sorted_index[0:10]]
```

```
array(['красив', 'лучш', 'блят', 'пиздец', 'мил', 'любл', 'прекрасн',
       'крут', 'хорош', 'нах'], dtype=object)
```

Рисунок 7 – Полученные показатели модели после подбора параметра максимальной глубины случайного леса

6) В пункте 3 вместо стемминга осуществите лемматизацию и проделайте пункты 3-5 с учетом другого типа подготовки данных.

Лемматизация осуществлялась с помощью библитеки pymorphy3, а именно модуля Mystem, который поддерживает контекстную лемматизацию.

```
# Mystem - контекстный лемматизатор + pip instal
m = Mystem()

x_test_lem = []
x_train_lem = []

# lemmatize
for i, t in df_lem_train.iterrows():
    lemmas = m.lemmatize(t[0])
    x_train_lem.append("".join(lemmas).strip())

for i, t in df_lem_test.iterrows():
    lemmas = m.lemmatize(t[0])
    x_test_lem.append("".join(lemmas).strip())
```

Рисунок 8 – Лемматизация текста

```
vec = TfidfVectorizer(stop_words=None)

# tf-idf test and train
matr_train_lem = vec.fit_transform(x_train_lem)
matr_test_lem = vec.transform(x_test_lem)
```

Рисунок 9 – Получение tf-idf после лемматизации

По аналогии со стеммингом модель с параметрами по умолчанию переобучена, поэтому также подберем параметры для каждой модели.

```
model_logreg = LogisticRegression(C=0.21)
model_logreg.fit(matr_train_lem, df_lem_train['label'])
y_logreg = model_logreg.predict(matr_train_lem)

accuracy = accuracy_score(df_lem_train['label'], y_logreg)
print(accuracy)
```

```
0.7255416940249507
```

```
model_logreg = LogisticRegression(C=0.21)
model_logreg.fit(matr_train_lem, df_lem_train['label'])
y_logreg = model_logreg.predict(matr_test_lem)

accuracy = accuracy_score(df_lem_test['label'], y_logreg)
print(accuracy)
```

```
0.7086614173228346
```

```
importance = model_logreg.coef_
abs_importance = abs(importance)
sorted_index = abs_importance.argsort()
sorted_index = sorted_index[0][::-1]
# топ-20 значимых слов
np.array(vec.get_feature_names_out())[sorted_index[0:10]]
```

```
array(['любить', 'блять', 'хороший', 'красивый', 'пиздец', 'вау', 'милый',
       'умирать', 'нравиться', 'сука'], dtype=object)
```

Рисунок 10 – параметры модели после подбора параметра логистической регрессии при лемматизации

```
model_random_forest = RandomForestClassifier(max_depth=25)
model_random_forest.fit(matr_train_lem, df_lem_train['label'])
y_forest = model_random_forest.predict(matr_test_lem)

accuracy = accuracy_score(df_lem_test['label'], y_forest)
print(accuracy)
```

```
0.6745406824146981
```

```
model_random_forest = RandomForestClassifier(max_depth=25)
model_random_forest.fit(matr_train_lem, df_lem_train['label'])
y_forest = model_random_forest.predict(matr_test_lem)

accuracy = accuracy_score(df_lem_test['label'], y_forest)
print(accuracy)
```

```
0.6640419947506562
```

```
importance = model_random_forest.feature_importances_
abs_importance = abs(importance)
sorted_index = abs_importance.argsort()[::-1]
# топ-10 значимых слов
np.array(vec.get_feature_names_out())[sorted_index[0:10]]
```

```
array(['любить', 'блять', 'хороший', 'красивый', 'вау', 'пиздец', 'милый',
       'умирать', 'нахуй', 'сука'], dtype=object)
```

Рисунок 11 – Полученные показатели модели после подбора параметра максимальной глубины случайного леса при лемматизации

7) Сравните результаты по качеству и по наиболее важным признакам (словам) между 2 обученными вариантами.

Таблица 1 – Сравнение стемминга и лемматизации

	Стемминг		Лемматизация	
	Логистическая	Случаный лес	Логистическая	Случаный лес
Accuracy	0.71	0.66	0.71	0.66
Топ-10 слов	**'любл',** **'красив',** **'блят',** **'пиздец',** **'мил',** 'крут', **'ва',** 'прекрасн', 'лучш', **'хорош'**	**'красив',** 'лучш', **'блят',** **'пиздец',** **'мил',** **'любл',** 'прекрасн', 'крут', **'хорош',** **'нах'**	**'любить',** **'блять',** **'хороший',** **'красивый',** **'пиздец',** **'вау',** **'милый',** 'умирать', 'нравиться', 'сука'	**'любить',** **'блять',** **'хороший',** **'красивый',** 'вау', **'пиздец',** **'милый',** 'умирать', **'нахуй',** 'сука'

Качество моделей (по метрике accuracy) абсолютно идентичное между лемматизацией и стеммингом. Если говорить про слова – то можно заметить, что в логичстической регрессии различие лишь в трех словах из первой десятки, причем с примерным сохранением порядка; в модели случайного леса – также различие лишь в трех словах, но порядок сохранился хуже.

Если же говорить о различие между самими моделями, то различие в полученной метрике незначительно (0.05).

Таким образом, можно сказать, что глобальных различий между лемматизацией и стеммингом не обнаружилось, а это могло произойти по следующим причинам:

- Не очень большой объем текста
- Неважность грамматической информации (для чего и нужно использовать лемматизацию)

Вывод – в ходе выполнения лабораторной работы я научился делать предобработку (очистку) текста, осуществлять лемматизацию, стемминг и нахождение tf-idf на языке python. На основе tf-idf получена модели логистической регрессии и случайного леса, которые в последствии были сравнены для 2 случаев.