МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ИТ

ОТЧЕТ

по производственной практике

Тема:

«Изучение и сравнение методов тональности твитов»

Студенты гр. [redacted] _____ Бочаров Г.С.
Сидоров Е.Г.

Руководитель _____ Васильев Н.В.

Санкт-Петербург
2021

ЗАДАНИЕ

НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ

Бочаров Г.С.
Сидоров Е.Г.

Группа[redacted]
Тема практики: изучение и сравнение методов тональности твитов

Задание на практику:

Построение модели, которая по тексту поста предсказывает его эмоциональную окраску. Изучение основных методов представления и векторизации текстов.

Сроки прохождения практики:

Дата сдачи отчета:

Дата защиты отчета:

Студенты _____ Бочаров Г.С.
Сидоров Е.Г.

Руководитель _____ Васильев Н.В.

АННОТАЦИЯ

Данная работа представляет собой исследование способов построения модели, предсказывающую тональность текста. Результатом работы является построение моделей на основе различных методов представления и векторизации текста. Также, для каждой модели измеряются ее параметры и производится выбор наилучшей модели среди построенных.

SUMMARY

This work is a study of ways to build a model that predicts the tonality of a text. The result of the work is the construction of models based on various methods of text representation and vectoring. Also, for each model, its parameters are measured and the best model is selected among the constructed ones.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью практики является построение модели, которая по тексту поста предсказывает его эмоциональную окраску, а также изучить основные методы обработки, представления и векторизации данных. Требуется построить разные модели используя разные методы представления и векторизации текстов (BagOFWords – мещок слов, TF-IDF и метод классификации на основе логистической регрессии), сравнить их эффективность используя macro average f1-scor как основную метрику качества. Основываясь на данных, полученных в результате тестирования необходимо выбрать наилучший метод вектори зации.

1. ХОД РАБОТЫ

1.1. Загрузка датасета.

Предоставленный набор данных, содержащий предложения и их эмоциональную окраску загружается с помощью средств библиотеки pandas. Из датасета удаляются столбцы, не рассматриваемые в процессе анализа текса и формируется сбалансированная выборка содержащая одинаковое количество положительных и отрицательных сообщений.

1.2. Предварительная обработка датасета.

Средствами библиотеки re из текста удаляюся слова, не являющиеся русскими, цифры, имена пользователей, хэштеги, знаки препинания и цифры. Производится замена заглавных букв на строчные, и ё на е.

При помощи библиотеки pymorphy2 производится удаление стоп-слов и лемматизация текста. Это позволяет умменьшить кол-во уникальных слов и упростить их понимание при обучении.

Производится разбиение на n-граммы.

Полученный датасет сохраняется в файл, так как время на обработку довольно велико.

1.3. Векторизация.

Данные разбиваюся на обучающую и тестовую выборки. При помощи библиотеки sklearn, производится векторизация слов двумя методами.

1.4. Построение модели и оценка качества.

Производится обучение модели логистической регресси. Оценивается качество предсказаний полученной модели на тестовой выборке. В качестве метрики качества используется f1_score.

2. *ВЕКТОРИЗАЦИЯ*

Векторизации — преобразование представления слов в числовые векторы для упрощения работы с ними. Для такой трансформации используются специальные модели. В данной работе рассмотрены методы векторизации такие как: BagOFWords – мещок слов, TF-IDF с разбиением текста на униграммы, биграммы и триграммы для каждого метода.

2.1. Мешок слов

Мешок слов — упрощенное представление текста, которое используется в обработке естественных языков и информационном поиске. В этой модели текст (одно предложение или весь документ) представляется в виде мешка (мультимножества) его слов без какого-либо учета грамматики и порядка слов, но с сохранением информации об их количестве.

На практике bag of words реализуется следующим образом: создается вектор длиной в словарь, для каждого слова считается количество вхождений в текст и это число подставляется на соответствующую позицию в векторе. Однако, при этом теряется порядок слов в тексте , а значит учитывается только влияние отдельных слов на окрас предложения, но не и комбинаций. Например, предложения “i have no cats” и “no, i have cats” будут идентичны в данном представлении, хотя их смыслы противоположны.

2.2. n-граммы

Целью построения N-граммных моделей является определение вероятности употребления заданной последовательности слов. Эта модель определяет и сохраняет смежные последовательности слов в тексте следовательно учитываает не только влияние отдельных слов на окрас но и их взаимное расположение.

2.3. TF-IDF

TF-IDF— статистическая мера, используемая для оценки важности слова в контексте , ядокументавляющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

Мера TF-IDF часто используется для представления документов коллекции в виде числовых векторов, отражающих важность использования каждого слова из некоторого набора слов (количество слов набора определяет размерность вектора) в каждом документе.

Таким образом, вес слова встресающегося во всем наборе данных слишком часто будет меньше.

3. МОДЕЛЬ ЛОГИСТИЧЕСКОЙ РЕГРЕССИИ

3.1. Основная идея логистической регрессии

В отличие от обычной регрессии, в методе логистической регрессии не производится предсказание значения числовой переменной исходя из выборки исходных значений. Вместо этого, значением функции является вероятность того, что данное исходное значение принадлежит к определенному классу. Для простоты, давайте предположим, что у нас есть только два класса и вероятность, которую мы будем определять, P_+вероятности того, что некоторое значение принадлежит классу "+". И конечно $P_- = 1 - P_+$. Таким образом, результат логистической регрессии всегда находится в интервале [0, 1].

Основная идея логистической регрессии заключается в том, что пространство исходных значений может быть разделено линейной границей (т.е. прямой) на две соответствующих классам области. Итак, что же имеется ввиду под линейной границей? В случае двух измерений — это просто прямая линия без изгибов. В случае трех— плоскость, и так далее. Эта граница задается в зависимости от имеющихся исходных данных и обучающего алгоритма. Чтобы все работало, точки исходных данных должны разделяться линейной границей на две вышеупомянутых области. Если точки исходных данных удовлетворяют этому требованию, то их можно назвать линейно разделяемыми.

3.2. Математическая основа логистической регрессии

Итак, как уже было сказано, в логит регрессионной модели предсказанные значения зависимой переменной или переменной отклика не могут быть меньше (или равными) 0, или больше (или равными) 1, не зависимо от значений независимых переменных; поэтому, эта модель часто используется для анализа бинарных зависимых переменных или переменных отклика.
При этом используется следующее уравнение регрессии

$$y=\exp(b_0+b_1*x_1+...+b_n*x_n)/[1+\exp(b_0+b_1*x_1+...+b_n*x_n)]$$

Легко увидеть, что независимо от регрессионных коэффициентов или величин x, предсказанные значения (y) в этой модели всегда будут лежать в диапазоне от 0 до 1.

Термин логит произошел от того, что эту модель легко линеаризовать с помощью логит преобразования. Предположим, что бинарная зависимая переменная y является непрерывной вероятностью p, лежащей в диапазоне от 0 до 1. Тогда можно преобразовать эту вероятность p следующим образом:

$$p' = \log_e \{p/(1-p)\}$$

Это преобразование называется логит или логистическим преобразованием.

Заметим, что p' теоретически может принимать любые значения от минус до плюс бесконечности. Поскольку логит преобразование решает проблему 0/1 границ для исходной зависимой переменной (вероятности), то можно использовать эти (логит преобразованные) значения в обычном линейном уравнении регресии.

Фактически, при проведении логит преобразования обеих частей логит регрессионного уравнения, приведенного выше, мы получим стандартную линейную модель множественной регрессии:

$$p' = b_0 + b_1*x_1 + b_2*x_2 + ... + b_n*x_n$$

Подобное уравнение нам уже знакомо. Решив его, мы получим значения регрессионных коэффициентов, по которым затем можно восстановить вероятность p.

4. АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

В результате работы, были получены оценки работы 6 способов построения модели на выборках разной длинны. Результа отображены в таблице 1.

	Bow, униграммы	Bow, биграммы	Bow, триграммы	TF-IDF, униграммы	TF-IDF, биграммы	TF-IDF, тригаммы
f1_score, size = 22384	0.9710537570711433	0.9710537570711433	0.9736301700811751	0.9708295319780486	0.9730667290891105	0.9731815852875907
f1_score, size = 44768	0.9718397662923548	0.9733512855005262	0.9732934683307284	0.9722834635290892	0.9732916007861334	0.9737951254199284
f1_score, size = 67152	0.9740742473877185	0.9732905064173244	0.9747100054129817	0.9742975630480464	0.9743721538472205	0.9742231547416813
f1_score, size = 89536	0.9739907239475367	0.9740473433396472	0.9743827085148449	0.9746048620417467	0.9741291145935311	0.9746054703356213
f1_score, size = 111920	0.9745687014441425	0.9738508762157287	0.9742989270453015	0.9746775207624657	0.9754601044697506	0.9745211520609379
f1_score, size = 134304	0.9772258694803011	0.9754548848739273	0.9754545673936799	0.9745765963902547	0.9750961661129256	0.9745208351279107
f1_score, size = 156688	0.9756259755825234	0.9752740050286275	0.9757060912708205	0.975383740148247	0.9750961661129256	0.9749532178990999
f1_score, size = 179072	0.975278220351918	0.9759068114184597	0.976201013558821	0.9752060244358932	0.9746043530034396	0.9747867401763026
f1_score, size = 201456	0.9756416812639956	0.9760391369813908	0.9751443413313836	0.9757390191096107	0.9757142696109984	0.9750061717916277
f1_score, size = 223840	0.9765146214180973	0.9754170976442098	0.9759436799589865	0.9750911656040018	0.9757286532368062	0.9750357731421855

ТАБЛИЦА 1

Из результатов Таблицы Видно, что метод Bow выигрывает у метода TF-idf примерно 0.001%, но метод Tf-iDf работает существенно быстрее на больших данных. Таким образом, можно установить, что метод Tf-idf работает лучше на данной выборке.

ЗАКЛЮЧЕНИЕ

Кратко подвести итоги, проанализировать соответствие поставленной цели и полученного результата.

В ходе работы были изучены и применены на практике разные методы анализа тональности текста. Была освоена работа с библиотеками для обработки данных. Изучен принцип работы основных методов векторизации.

Для построенных моделей проведено сравнение по их числовой характеристики f1-score, однако по данной характеристике модели отличаются не сильно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.https://habr.com/ru/company/ods/blog/322626/

2. https://medium.com/@bigdataschool/

3.https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%88%D0%BE%D0%BA_%D1%81%D0%BB%D0%BE%D0%B2

4. https://ru.wikipedia.org/wiki/TF-IDF

5. https://ru.stackoverflow.com/

6. https://m.habr.com/ru/company/mailru/blog/417767/

7. https://pymorphy2.readthedocs.io/en/0.2/user/index.html

8.https://datastart.ru/blog/read/plavnoe-vvedenie-v-natural-language-processing-nlp

9. Лутц Марк Изучаем Python. Том 1

10. https://russianblogs.com/article/1795892492/

11. http://statistica.ru/theory/logisticheskaya-regressiya/

12. https://habr.com/ru/company/io/blog/265007/

ПРИЛОЖЕНИЕ А
НАЗВАНИЕ ПРИЛОЖЕНИЯ

```
import pandas as pd
import numpy as np
import re
import os
import sys
import sqlite3
import nltk
#nltk.download('punkt')
#nltk.download('stopwords')
from sklearn.model_selection import train_test_split
import logging
import multiprocessing
import gensim
from gensim.models import Word2Vec
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import classification_report
import joblib
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.util import ngrams
#from IPython.display import display
import pymorphy2
from sklearn import *
from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.metrics import f1_score
morph = pymorphy2.MorphAnalyzer(lang = 'ru')
stop_words = stopwords.words('russian')


# предобработка текста, удаление стоп-слов, токенизация, лемматизация
def preprocess_text(text, grams):

    text = text.lower().replace("ё", "е")
    text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', ' ', text)
```

```
    text = re.sub('[^a-яА-Я()]+', ' ', text)

    text = re.sub('@[^\s]+', ' ', text)
    text = re.sub('#[^\s]+', ' ', text)

    # токенизация
    tokens = nltk.word_tokenize(text)
    #удаление стоп слов
    text = [word for word in tokens if word not in stop_words ]


    #лемматизация
    text = [morph.parse(w)[0].normal_form for w in  text]

    if grams == 1 :
       text = ' '.join(text)
    else :
       text = list(ngrams(text, grams))
       text = [' '.join(w) for w in text]
       text = ' '.join(text)


    return text




# загрузка обработанных данных в датасет


try:
    data_unigram = pd.read_csv("unigram_preprocessed.csv", encoding ='utf8')
    data_bigram = pd.read_csv("bigram_preprocessed.csv", encoding ='utf8')
    data_threegram = pd.read_csv("threegram_preprocessed.csv", encoding ='utf8')
except FileNotFoundError:

    n = ['1', '2', 'user', 'text', 'value', '6', '7', '8', '9', '10', '11',
'12']
    data_positive = pd.read_csv('datasets/positive.csv', encoding ='utf8',
sep=';', names =["1","2","3","text","value","6","7","8","9","10","11","12"],
usecols=['text'] )
```

```
    data_negative = pd.read_csv('datasets/negative.csv', encoding ='utf8',
sep=';', names =["1","2","3","text","value","6","7","8","9","10","11","12"],
usecols=['text'])
    sample_size = min(data_positive.shape[0], data_negative.shape[0])
    raw_data = np.concatenate((data_positive['text'].values[:sample_size],
data_negative['text'].values[:sample_size]), axis=0)

    labels = [1] * sample_size + [0] * sample_size   # список окрасов
    data_unigram = pd.DataFrame(data=None, index=None,
columns={"review" ,"label", "preprocessed_review"}, dtype=None, copy=False)
    data_unigram['review'] = raw_data
    data_unigram['label'] = labels
    data_unigram.to_csv("prep1.csv", index = False)
    prep1 = pd.read_csv("prep1.csv", chunksize = 10000)

    data_bigram = pd.DataFrame(data=None, index=None,
columns={"review" ,"label", "preprocessed_review"}, dtype=None, copy=False)
    data_bigram['review'] = raw_data
    data_bigram['label'] = labels
    data_bigram.to_csv("prep2.csv", index = False)
    prep2 = pd.read_csv("prep2.csv", chunksize = 10000)

    data_threegram = pd.DataFrame(data=None, index=None,
columns={"review" ,"label", "preprocessed_review"}, dtype=None, copy=False)
    data_threegram['review'] = raw_data
    data_threegram['label'] = labels
    data_threegram.to_csv("prep3.csv", index = False)
    prep3 = pd.read_csv("prep3.csv", chunksize = 10000)

    chunk_list1 = []
    chunk_list2 = []
    chunk_list3 = []

    for data_chunk in prep1:
       filteredchunk1 = data_chunk['review'].apply(lambda review :
preprocess_text(review ,1) )
       chunk_list1.append(filteredchunk1)

    d_u= pd.concat(chunk_list1)
    data_unigram['preprocessed_review']  = d_u.to_frame()
    data_unigram.to_csv("unigram_preprocessed.csv", index = False)
```

```
    for data_chunk in prep2:
       filteredchunk2 = data_chunk['review'].apply(lambda review :
preprocess_text(review ,1) )
       chunk_list2.append(filteredchunk2)

    d_b= pd.concat(chunk_list2)
    data_bigram['preprocessed_review']  = d_b.to_frame()
    data_bigram.to_csv("bigram_preprocessed.csv", index = False)


    for data_chunk in prep3:
       filteredchunk3 = data_chunk['review'].apply(lambda review :
preprocess_text(review ,1) )
       chunk_list3.append(filteredchunk3)


    d_t= pd.concat(chunk_list3)
    data_threegram['preprocessed_review']  = d_t.to_frame()
    data_threegram.to_csv("threegram_preprocessed.csv", index = False)

sample_size = len(data_unigram.index)
print(sample_size)




# загрузка модели

def load_model(filename):
    return joblib.load(filename)

# сохранение модели
def save_model(filename, model):
    joblib.dump(model, filename)


    # векторизация методом мешка слов и построение модели
def model_bow(data, grams):
    d = data.copy()
    y = d['label'].values
    d.drop(['label'] , axis = 1, inplace = True)
```

```
    x_train, x_test, y_train, y_test = train_test_split(d, y, test_size=0.4,
stratify = y, random_state = 0)
    vect = CountVectorizer(token_pattern=r"(?u)\b\w\w+\b|!|\?|\"|\'")
    X_train_review_bow =
vect.fit_transform(x_train['preprocessed_review'].values.astype('U'))
    X_test_review_bow =
vect.transform(x_test['preprocessed_review'].values.astype('U'))

    print(len(vect.get_feature_names()))

    clf = LogisticRegression(solver='sag', max_iter=100000)

    clf.fit(X_train_review_bow, y_train)

    y_pred = clf.predict(X_test_review_bow)

    print('f1_score : ', f1_score(y_test, y_pred, average='weighted'))

    # векторизация методом tf-idf и построение модели
def model_tfidf(data, grams):

    d = data.copy()
    y = d['label'].values
    d.drop(['label'] , axis = 1, inplace = True)
    x_train, x_test, y_train, y_test = train_test_split(d, y, test_size=0.4,
stratify = y)
    vectorizer = TfidfVectorizer()
    X_train_review_bow =
vectorizer.fit_transform(x_train['preprocessed_review'].values.astype('U'))
    X_test_review_bow =
vectorizer.transform(x_test['preprocessed_review'].values.astype('U'))
    clf = LogisticRegression(solver='sag', max_iter=10000)
    clf.fit(X_train_review_bow, y_train)
    y_pred = clf.predict(X_test_review_bow)
    print('Tf-idf', grams , '-grams :\n',  classification_report(y_test,
y_pred))


#Тесты на датафреймах разной длины
def model_bow_T(data, grams, i = 0, j = 0):
```

```
    d = data.copy()[sample_size//2 - (i+1)*(sample_size//2//j)  :
sample_size//2+ (i+1)*(sample_size//2//j)]

    y = d['label'].values

    d.drop(['label'] , axis = 1, inplace = True)

    x_train, x_test, y_train, y_test = train_test_split(d, y, test_size=0.4,
stratify = y)
    vect = CountVectorizer(token_pattern=r"(?u)\b\w\w+\b|!|\?|\(|\)")


    X_train_review_bow =
vect.fit_transform(x_train['preprocessed_review'].values.astype('U'))
    X_test_review_bow =
vect.transform(x_test['preprocessed_review'].values.astype('U'))


    clf = LogisticRegression(solver='sag', max_iter=10000)

    clf.fit(X_train_review_bow, y_train)

    y_pred = clf.predict(X_test_review_bow)
    print('Test N ', i, ' Size = ', 2*((i+1)*(sample_size//2//j)))
    print('f1_score : ', f1_score(y_test, y_pred, average='macro'))

    print('Feature_size: ',len(vect.get_feature_names()))

def model_tfidf_T(data, grams, i = 0, j = 0):


    d = data.copy()[sample_size//2 - (i+1)*(sample_size//2//j)  :
sample_size//2+ (i+1)*(sample_size//2//j)]

    y = d['label'].values

    d.drop(['label'] , axis = 1, inplace = True)

    x_train, x_test, y_train, y_test = train_test_split(d, y, test_size=0.4,
stratify = y)

    vect = TfidfVectorizer(token_pattern=r"(?u)\b\w\w+\b|!|\?|\(|\)")
```

```
    X_train_review_bow =
vect.fit_transform(x_train['preprocessed_review'].values.astype('U'))
    X_test_review_bow =
vect.transform(x_test['preprocessed_review'].values.astype('U'))



    clf = LogisticRegression(solver='sag', max_iter=10000)

    clf.fit(X_train_review_bow, y_train)

    y_pred = clf.predict(X_test_review_bow)
    print('Test N ', i, ' Size = ', 2*((i+1)*(sample_size//2//j)))
    print('f1_score : ', f1_score(y_test, y_pred, average='macro'))

    print('Feature_size: ',len(vect.get_feature_names()))


print('\\\\--------B_O_W--------////')
print()
print('\\\\--------UNIGRAM--------////')


# вывод результатов

for i in range(10):
    model_bow_T(data_unigram, 1, i, 10)
print('\\\\--------BIGRAM--------////')
for i in range(10):
    model_bow_T(data_bigram, 2, i, 10)

print('\\\\--------TRIGRAM--------////')
for i in range(10):
    model_bow_T(data_threegram, 3, i, 10)


print('\\\\\\\\\\\\\\\\|////////////////')
print('////////////////|\\\\\\\\\\\\\\\\')

print()
print('\\\\--------T_F_I_D_F--------////')
```

```
print()

print()

print('\\\\--------UNIGRAM--------////')
for i in range(10):
    model_tfidf_T(data_unigram, 1, i, 10)

print('\\\\--------BIGRAM--------////')
for i in range(10):
    model_tfidf_T(data_bigram, 2, i, 10)

print('\\\\--------TRIGRAM--------////')
for i in range(10):
    model_tfidf_T(data_threegram, 3, i, 10)
```