Постановка задачи

В качестве задачи был выбран анализ тональности текста целиком. На вход подается текст, необходимо определить положительную или отрицательную оценку он имеет. Таким образом, рассматривалась задача бинарной классификации.

В настоящее время задачи определения эмоциональной оценки текста и извлечения мнений пользователей являются одними из самых популярных в области обработки текста. Они нашли применение во многих сферах, например: анализ восприятия товаров и услуг, мониторинг блогов, политические исследования, киноиндустрия, бизнес, сбор мнений о компаниях и т. д. На актуальность этого вопроса в основном повлияло увеличение интернет-пользователей и развитие сайтов, на которых они могли бы обсуждать те или иные объекты.

Существует множество различных работ, посвященных тональному анализу. Большинство из низ использует либо машинное обучение, либо словари и набор правил. Первые из них в основном обучаются на заранее размеченных текстах и используют полученную модель при дальнейшей работе. Для обучения текст представляется в виде вектора признаков. В целом, эти методы справляются с поставленной задачей, но модели работают на конкретных предметных областях и плохо переносятся на другие. Также размеченные коллекции для обучения требуют больших затрат, так как получение достаточных для хороших результатов объемов занимает длительное время у экспертов. Не все методы машинного обучения, например нейронные сети, хорошо интерпретируются, поэтому полученные ошибки трудно исправить. Методы, основанные на словарях и правилах, используют словарь оценочной лексики. Его можно получить с помощью ручной или автоматической разметки. Иногда переводят уже существующие словари с других языков, но без учета значений слов качество таких словарей низкое. При ручной разметке возникает такая же проблема, что и в обучении с учителем. Так или иначе результаты зависят от полученных словарей. Правила, использующиеся для получения итоговой эмоциональной окраски, похожи на признаки в первом подходе, так как в каждом исследовании берутся разные, но есть и набор общих. В отличие от первого подхода, результаты хорошо объясняются, что помогает дорабатывать и улучшать методы с учетом выявленных ошибок.

Основными мерами качества в данной задаче являются правильность классификации (accuracy) и $F1$-мера в варианте макро- или микро-усреднения.

Обзор решений

Были рассмотрены следующие системы: Texterra API, Eureka Engine и Alchemy API. Первая была разработана в ИСП РАН и предоставляет REST API к реализованным методам. Также есть веб-интерфейс. Помимо анализа тональности она решает ряд других задач: морфологический и синтаксический анализ, выделение именованных сущностей и т. д. Языком реализации является Java. Использует Apache OpenNLP и Yandex MyStem для решения некоторых задач. Тональность определяется с помощью метода опорных векторов. Опубликованные результаты для русского языка: $F1 = 0.86$ и $F1 = 0.91$ для присутствия и полярности (положительная, отрицательная и нейтральная) оценки соответственно.

Следующая система, Eureka Engine, также решает различные задачи обработки текстов: определение языка, тональности (объектное и безобъектное), классификация текстов, определение и классификация именованных сущностей, морфологический анализ. Программу можно купить или попробовать через веб-интерфейс или REST API. Тоже выделяется три класса тональности, но есть возможность определить отношение к конкретному объекту. Объектом может быть как отдельное слово, так и словосочетание. Результаты показали 86-процентную точность в среднем по трем классам.

Последним инструментом является Alchemy API. Как следует из названия предоставляется REST API, и как и в остальных системах — веб-интерфейс. Принадлежит компании IBM. Умеет извлекать ключевые слова, именованные сущности и отношения между последними. Тональный анализ проводится как к конкретному объекту, так и общий по тексту. В качестве алгоритмов используются машинное и глубинное обучения. Результаты не опубликованы. Более подробное описание приведено в таблице 1.

название системы	Texterra API	Eureka Engine	Alchemy API
разработчики	ISP RAS	PalitrumLab	IBM
год создания	2014	2015	2005
ссылка на сайт	api.ispras.ru/demo/texterra	eurekaengine.ru	ibm.com/watson/alchemy-api.html
язык программирования	Java	неизвестно	неизвестно
виды интерфейсов (консоль, API)	REST API + Web Demo	REST API + Web Demo + можно купить	REST API + Web Demo
решаемые задачи	тональный, морфологический и синтаксический анализ; выделение именованных сущностей; исправление опечаток	определение языка, тональности (объектное и безобъектное); классификация текстов; определение и классификация именованных сущностей; морфологический анализ	тональный анализ (объектный и безобъектный); извлечение ключевых слов, именованных сущностей и отношениймежду последними
используемые подходы и методы	машинное обучение	машинное обучение	машинное обучение, глубинное обучение
язык обрабатываемых текстов	русский, английский	русский, английский	английский
тематика обрабатываемых текстов	любая	любая	любая

Таблица 1: Описание рассмотренных систем

Инструменты

NLTK

NLTK[1] — пакет библиотек и программ для символьной и статистической обработки естественного языка, написанных на языке программирования Python. Проект начал существование в 2001-м году и до сих пор развивается. Код распространяется под лицензией Apache 2.0. Некоторые задачи, которые может решить NLTK:

- сегментация на предложения;
- токенизация;
- морфологический анализ;
- анализ тональности;
- машинный перевод;
- построение дерева зависимостей;
- подсчет n-грамм.

Большинство из них решаются только для английского языка, но часть применима и для русского. Проект содержит обширную документацию, что позволяет с легкостью интегрировать его для решения своих задач по обработке естественного языка.

scikit-learn

Scikit-learn[2] — это библиотека для машинного обучения на языке программирования Python с открытым исходным кодом (лицензия BSD). Библиотека развивается с 2007-го года и по сей день. С помощью нее можно реализовать различные алгоритмы классификации, регрессии и кластеризации. Также есть много дополнительных инструментов, например: grid search, кросс-валидация, выбор признаков, предобработка данных и т. д. Помимо подробной документации с примерами, очень полезным является рисунок 1 с подсказками по выбору нужного алгоритма.

pymorphy2

pymorphy2[3] — морфологический анализатор с открытым исходным кодом (лицензия MIT), разработанный на языке программирования Python. Использует словарный подход. В качестве словаря выступает OpenCorpora[4]. Кроме извлечения грамматической информации о слове, pymorphy2 также

[1]Natural Language Toolkit. URL: http://www.nltk.org.

[2]scikit-learn: machine learning in Python. URL: http://scikit-learn.org/stable/.

[3]Морфологический анализатор pymorphy2. URL: http://pymorphy2.readthedocs.io/en/latest/.

[4]OpenCorpora: открытый корпус русского языка. URL: http://www.opencorpora.org.

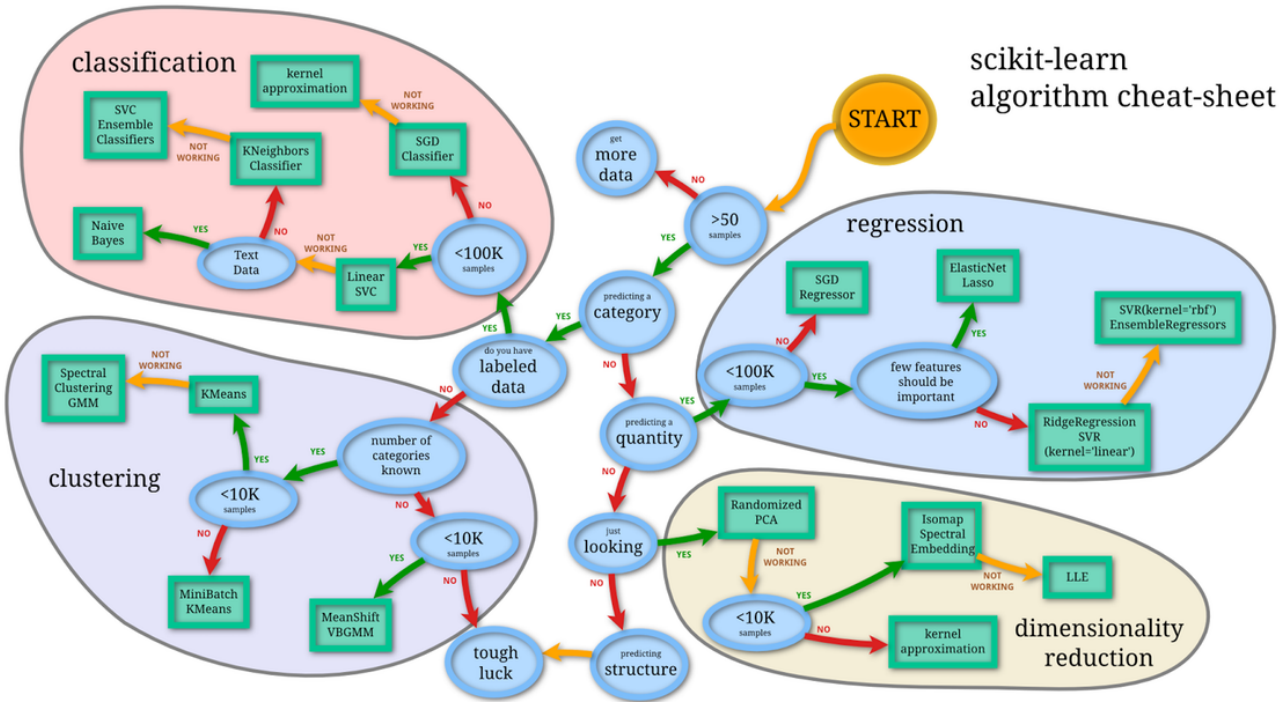

Рис. 1: Выбор нужного алгоритма

умеет ставить слово в нужную форму, например начальную. Алгоритм основывается на лингвистических правилах и может обрабатывать неизвестные слова. Последнее возможно за счет набора правил, например: отсечение известных префиксов, предсказание по концу слова, анализ составных слов, и т. д.

Данные

Рассматривались рецензии на фильмы с сайта КиноПоиск[1]. Так как рецензии имеют шкалу оценок от 1 до 10, то было необходимо перевести их в бинарный вид перед началом работы. Помимо оценок пользователи также помечают отзыв как положительный или отрицательный. Стоит отметить, что бывают рецензии, отмеченные как отрицательные, но их численная оценка может быть высокой. Поэтому перевод в бинарный вид производился следующим образом:

- положительная рецензия с оценкой от 7 до 10 включительно: положительно;
- отрицательная рецензия с оценкой от 1 до 4 включительно: отрицательно.

[1]КиноПоиск — Все фильмы планеты. URL: http://www.kinopoisk.ru.

Кол-во \ Рецензии	Положит.	Отрицат.	Все
Предложений	139090	156014	295104
Предложений в ср. на текст	28	31	30
Слов	1736404	1867523	3603927
Слов в ср. на предложение	12	12	12
Слов в ср. на текст	347	374	360

Таблица 2: Статистика рецензий

В результате было отобрано 10 тысяч рецензий, по 5 тысяч на каждый класс. Статистику по полученным рецензиям можно увидеть в таблице 2. Перед сбором статистики была произведена предобработка, описанная в следующем разделе.

Предобработка текста

В целом рецензии на КиноПоиске по структуре и грамотности ближе к новостным сообщениям, чем к текстам из социальны сетей, но небольшая предобработка все рано необходима. Производились такие действия, как:

- удаление ссылок;
- удаление смайлов;
- удаление переносов строк;
- замена повторяющихся точек, пробелов или табуляций на один символ.

Результаты

Для решения задачи был рассмотрен ряд классификаторов из библиотеки scikit-learn. В качестве базового решения был взят метод опорных векторов[1] с униграммами и биграммами в виде характеристик и радиальным ядром. Для их извлечения использовался класс `CountVectorizer` из библиотеки NLTK. При этом предобработка текста не производилась.

Остальные решения использовали для характеристик униграммы и биграммы как для слов, так и для символов. Для нормализации в соответствии с мерой *tf-idf* был взят класс `TfidfTransformer` с нормой $l2$. Классу `CountVectorizer` в параметре передавался реализованный класс `LemmaTokenizer`, который использует pymorphy2 для постановки слова в начальную форму. Предобработка выполнялась в соответствии с предыдущим разделом. Параметры для классификаторов можно увидеть в коде программы в приложении А.

[1]Support vector machine. URL: http://en.wikipedia.org/wiki/Support_vector_machine.

Оценка производилась с помощью кросс-валидации при $k = 10$, которая сохраняет процентное соотношение для каждого класса (класс `StratifiedKFold`). Результаты показаны в таблицах 3, 4. В первой из них значения макро-усреднены для двух классов и по 10 запускам, поэтому некоторые значения меры F1 могут не совпадать со значением из формулы $\frac{2 \cdot PR \cdot RE}{PR+RE}$. Все классификаторы обошли базовый, а `Perceptron`[1] и `MultinomialNB`[2] превысили отметку в 90% для меры F1, что является хорошим результатом. Для улучшения результатов необходим более тщательный подбор признаков и параметров.

Классификатор	**Точность**	**Полнота**	**F1**
SVC	62.89	52.76	41.06
RandomForestClassifier	63.52	63.83	62.84
LogisticRegression	87.41	86.89	86.84
Perceptron	93.9	93.86	93.86
PassiveAggressiveClassifier	92.56	93.01	88.96
MultinomialNB	94.18	94.07	94.06

Таблица 3: Макро-усреднение результатов кросс-валидации различных классификаторов

Классификатор	**Класс**	**Точность**	**Полнота**	**F1**
SVC	отриц.	74.31	8.3	14.82
	полож.	51.47	97.22	67.3
RandomForestClassifier	отриц.	65.69	64.82	64.32
	полож.	64.77	64.32	63.57
LogisticRegression	отриц.	83.39	92.38	87.61
	полож.	91.44	81.4	86.06
Perceptron	отриц.	93.2	94.66	93.91
	полож.	94.6	93.06	93.81
PassiveAggressiveClassifier	отриц.	88.97	88.5	85.46
	полож.	92.46	86.84	88.17
MultinomialNB	отриц.	92.98	95.5	94.18
	полож.	95.38	92.64	93.95

Таблица 4: Результаты кросс-валидации по классам

[1]Perceptron. URL: http://en.wikipedia.org/wiki/Perceptron.

[2]Naive Bayes classifier. URL: http://en.wikipedia.org/wiki/Naive_Bayes_classifier.

Пример правильно определенной положительной рецензии.

Только что закончил просмотр этого Фильма... Попытаюсь описать свои ощущения.

Это неповторимо... Блестяще... Несравненно... Мне сейчас сложно собрать свои мысли воедино, и сказать что-нибудь более или менее внятное, но я все же попытаюсь.

Я долго не решался посмотреть этот фильм. Возможно, из-за моего предубеждения насчет певиц в кино. Я всегда считал, что каждый человек должен заниматься СВОИМ делом, а не пытаться успеть везде. Но Бьорк стала исключением. Ее игра просто завораживает, не отпускает от экрана.

Элементы мюзикла, которыми украшен этот фильм, в первые минуты просмотра несколько раздражали. (Открою секрет: ну не поклонник я этого жанра в кинематографе, уж не судите строго.) Но потом, по мере развития сюжета, я начал с нетерпением ждать их. И уже не мог себе представить этот фильм без этих музыкальных номеров, без этих кружев, на полотне, сплетенном Ларсом Фон Триером.

Поражает и операторская работа. На протяжении всего фильма оператору удавалось выхватывать самую суть, квинтэссенцию сцен. Неожиданные ракурсы, переезды камеры с одного положения на другое, даже это проигрывание камерой создают особую атмосферу фильма.

Вот уж никогда раньше не думал, что в фильме мне понравится такая, с первого взгляда незаметная вещь, как монтаж. Но и он оказался на высоте! Как ни странно это может прозвучать, лично мне было очень интересно смотреть, чего же Ларс Фон Триер не покажет, что он захочет вырезать, убрать при монтаже за ненадобностью.

Фильм достоин всяческих похвал. (Открою ещё один секрет: это третий фильм за всю мою жизнь, во время просмотра которого, я пустил слезу... даже две.)

10 из 10.

Пример отрицательной рецензии, распознанной как положительная.

Эндрю (Дэйн Де Хаан) живёт с больной мамой и отцом, бывшим пожарником, а теперь инвалидом. На скопленные деньги он купил старенькую, но вполне рабочую видеокамеру и начал снимать всё, что с ним происходит дома, в школе, на вечеринке, то есть вести хронику своей жизни. Однажды после вечеринки приятели Мэйт и Стив пригласили его заснять таинственную пещеру. Друзья спустились вниз для съёмок и получили сильное облучение. На следующий день они обнаруживают у себя невероятные способности телекинеза.

На мой взгляд, основная идея фильма в том, что любой подаренный судьбой шанс можно использовать как во благо, так и во вред себе и окружающим. Если Мэйт и Стив применяют новую опцию лишь для развлечения, то Эндрю находит удовольствие в том, чтобы использовать силу в недобрых целях. Он ощущает себя сверхчеловеком и альфа-хищником. Новый дар проявил его сущность и Эндрю даже подводит философское обоснование своей жестокости.

Фильм снят с ручной камеры, чтобы усилить ощущение хоум видео намеренно используется трясущийся кадр, что изрядно раздражает. Игра актёров весьма посредственная, даже особо выделить некого. Спецэффекты, применённые в фильме, виданы уже не раз и даже сыграны в компьютерных играх. Скажу откровенно, я не отношусь к целевой аудитории подобных фильмов и потому моя оценка

4 из 10

A Код программы

```
import glob
import os
import re
import shutil
import ujson

import numpy as np
import pymorphy2
import requests
from joblib import Parallel
from joblib import delayed
from lxml import etree
from nltk import word_tokenize, sent_tokenize,
  RegexpTokenizer, defaultdict
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import
  CountVectorizer, TfidfTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import
  PassiveAggressiveClassifier
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn.metrics import
  precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.svm import SVC

BASE_URL = 'https://www.kinopoisk.ru'
BASE_USER_URL = BASE_URL + '/user/3927381/votes/list/
  ord/date/page/%d/#list'
FILMS_REVIEWS = '%s/ord/rating/perpage/200/page/%d/#
  list'
REVIEWS_DIR = './reviews'


def get_links(page_number):
    films_links = []
```

```
        doc = etree.HTML(requests.get(BASE_USER_URL %
          page_number).text)
        page_films_links = doc.xpath('.//div[@class="
          nameRus"]/a[not(@style)]/@href')
        if page_films_links:
            films_links.extend([BASE_URL + x for x in
              page_films_links])
            print(BASE_USER_URL % page_number)
        return films_links


def to_windows_1251(match):
    try:
        return bytes([ord(match.group(0))]).decode('
          windows-1251')
    except UnicodeDecodeError:
        return ''


def restore_windows_1252_characters(s):
    return re.sub(r'[\u0080-\u0099]', to_windows_1251,
      s)


def normalize(text):
    text = restore_windows_1252_characters(text)
    return text


def get_reviews(film_link):
    print(film_link)

    reviews = {'good': [], 'bad': []}
    i = 1
    while True:
        html = requests.get(FILMS_REVIEWS % (film_link,
           i)).text
        html = re.sub('<[brisupa /]*?>', '', html)
        html = re.sub('<a.*?>', '', html)
        doc = etree.HTML(html)
```

```
            good = doc.xpath('.//div[contains(@class, "good
              ")]//span[@itemprop="reviewBody"]')
            good_texts = doc.xpath('.//div[contains(@class,
               "good")]//span[@itemprop="reviewBody"]/text
              ()')
            if len(good) != len(good_texts):
                assert False
            if good:
                reviews['good'].extend([normalize(text) for
                    text in good_texts])

            bad = doc.xpath('.//div[contains(@class, "bad")
              ]//span[@itemprop="reviewBody"]')
            bad_texts = doc.xpath('.//div[contains(@class,
              "bad")]//span[@itemprop="reviewBody"]/text()
              ')
            if len(bad) != len(bad_texts):
                assert False
            if bad:
                reviews['bad'].extend([normalize(text) for
                  text in bad_texts])

            if not good and not bad:
                break
            else:
                i += 1
    return reviews


def check_review(review, interval):
    scores = re.findall(r'(\d+).*?\b??\b.*?\d+', review
      , re.IGNORECASE)
    if scores and interval[0] <= int(scores[-1]) <=
      interval[1]:
        return review
    else:
        return None


def filter_reviews(reviews, interval):
```

```
    results = Parallel(n_jobs=-1, verbose=10)(delayed(
      check_review)(review, interval) for review in
      reviews)
    return [result for result in results if result]


def chunks(l, n):
    for i in range(0, len(l), n):
        yield l[i:i + n]


def prepare(text):
    # links
    text = re.sub(r'http[s]?://(?:[a-zA-Za-яA-Я
      ]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a
      -fA-F]))+', '', text)
    # smiles
    text = re.sub(r'[\)]+', '.', text)
    text = re.sub(r'[\(]+', '.', text)
    # newlines
    text = re.sub(r'[\r\n]+', '. ', text)
    text = re.sub(r'[\.]+', '.', text)
    # tabs
    text = re.sub(r'\t+', ' ', text)
    text = re.sub(r' +', ' ', text)

    return text


class LemmaTokenizer(object):
    def __init__(self):
        self.morph = pymorphy2.MorphAnalyzer(
          result_type=None)
        self.cache = {}

    def process(self, token):
        value = self.cache.get(token)
        if not value:
            value = self.morph.normal_forms(token)[0]
            self.cache[token] = value
        return value
```

```
    def __call__(self, doc):
        return [self.process(token) for token in
          word_tokenize(doc)]


def scrap_reviews():
    if os.path.exists(REVIEWS_DIR):
        shutil.rmtree(REVIEWS_DIR)
    os.makedirs(REVIEWS_DIR)

    films_links = []
    results = Parallel(n_jobs=-1, verbose=10)(delayed(
      get_links)(page_number + 1) for page_number in
      range(50))
    for result in results:
        films_links.extend(result)
    print(len(films_links))

    reviews = {'good': [], 'bad': []}
    results = Parallel(n_jobs=-1, verbose=10)(delayed(
      get_reviews)(film_link) for film_link in
      films_links)
    for result in results:
        reviews['good'].extend(result['good'])
        reviews['bad'].extend(result['bad'])
    print(len(reviews['good']), len(reviews['bad']))

    intervals = {'good': [7, 10], 'bad': [1, 4]}
    for polarity, polarity_reviews in reviews.items():
        reviews[polarity] = filter_reviews(
          polarity_reviews, intervals[polarity])
    print(len(reviews['good']), len(reviews['bad']))

    final_json = []
    for polarity, polarity_reviews in reviews.items():
        for review in polarity_reviews:
            final_json.append({'text': review, 'p': 1
              if polarity == 'good' else 0})
```

```
        for idx, chunk in enumerate(chunks(final_json, 500)
          ):
            with open(REVIEWS_DIR + '/reviews_%d.json' % (
              idx + 1), mode='w', encoding='utf8') as file
              :
                file.write(ujson.dumps(chunk, indent=4,
                  ensure_ascii=False))

        reviews_files = glob.glob(REVIEWS_DIR + '/reviews_*
          ')
        reviews = []
        for reviews_file in reviews_files:
            with open(reviews_file, mode='r', encoding='
              utf8') as file:
                reviews.extend(ujson.load(file))

        try:
            with open('reviews.json', mode='w', encoding='
              utf8') as file:
                ujson.dump(reviews, file, ensure_ascii=
                  False)
        except MemoryError:
            pass


    def load_reviews(size=None):
        reviews = []
        if os.path.exists('reviews.json'):
            try:
                with open('reviews.json', mode='r',
                  encoding='utf8') as file:
                    reviews = ujson.load(file)
            except MemoryError:
                pass

        if not reviews:
            reviews_files = glob.glob(REVIEWS_DIR + '/
              reviews_*')
            reviews = []
            for reviews_file in reviews_files:
```

```
            with open(reviews_file, mode='r', encoding=
              'utf8') as file:
                reviews.extend(ujson.load(file))

    good_reviews = [review for review in reviews if
      review['p'] == 1]
    bad_reviews = [review for review in reviews if
      review['p'] == 0]

    if not size or size > len(reviews):
        size = len(reviews)

    return good_reviews[:size] \
           + bad_reviews[:size]

if __name__ == '__main__':
    reviews = load_reviews(5000)

    STATS = False
    if STATS:
        good_reviews = [prepare(review['text']) for
          review in reviews if review['p'] == 1]
        bad_reviews = [prepare(review['text']) for
          review in reviews if review['p'] == 0]

        # SENTENCES
        good_sentences_count = 0
        for review in good_reviews:
            good_sentences_count += len(sent_tokenize(
              review))
        print('good all sents: %d' %
          good_sentences_count)
        print('good avg sents: %d' % round(
          good_sentences_count / len(good_reviews)))

        bad_sentences_count = 0
        for review in bad_reviews:
            bad_sentences_count += len(sent_tokenize(
              review))
        print('bad all sents: %d' % bad_sentences_count
          )
```

```
print('bad avg sent: %d' % round(
  bad_sentences_count / len(bad_reviews)))

print('total all sents: %d' % (
  good_sentences_count + bad_sentences_count))
print('total avg sents: %d' % round((
  good_sentences_count + bad_sentences_count)
  / len(reviews)))
print()

# WORDS
tokenizer = RegexpTokenizer(r'[a-zA-Z?-??-?0-9_
  -]+')

good_words = []
for review in good_reviews:
    good_words.extend(tokenizer.tokenize(review
      ))
print('good all words: %d' % len(good_words))
print('good avg words per sent: %d' % round(len
  (good_words) / good_sentences_count))
print('good avg words per text: %d' % round(len
  (good_words) / len(good_reviews)))

bad_words = []
for review in bad_reviews:
    bad_words.extend(tokenizer.tokenize(review)
      )
print('bad all words: %d' % len(bad_words))
print('bad avg words per sent: %d' % round(len(
  bad_words) / bad_sentences_count))
print('bad avg words per text: %d' % round(len(
  bad_words) / len(bad_reviews)))

print('total all words: %d' % len(good_words +
  bad_words))
print('total avg words per sent: %d'
      % round(len(good_words + bad_words) / (
        good_sentences_count +
        bad_sentences_count)))
```

```
        print('total avg words per text: %d' % round(
          len(good_words + bad_words) / len(reviews)))
        print()
        exit(0)

X = np.array([review['text'] for review in reviews
  ])
y = np.array([review['p'] for review in reviews])

TEST = False
pipelines = []
if TEST:
    vectorizer = CountVectorizer(analyzer='word',
      ngram_range=(1, 2))
    pipeline = Pipeline([
        ('features', vectorizer),
        ('clf', SVC())
    ])

    pipelines.append(pipeline)
else:
    X = np.array([prepare(x) for x in X])
    tokenizer = LemmaTokenizer()

    clfs = [
        RandomForestClassifier(max_depth=3, n_jobs
          =-1, n_estimators=5),
        PassiveAggressiveClassifier(n_iter=10,
          n_jobs=-1),
        Perceptron(n_iter=10, alpha=0.00001, n_jobs
          =-1),
        LogisticRegression(C=3, solver='liblinear',
           dual=True, class_weight='balanced',
          n_jobs=-1),
        MultinomialNB(),
    ]

    estimators = [
        ('CV1', CountVectorizer(analyzer='word',
          ngram_range=(1, 2), tokenizer=tokenizer)
          ),
```

```
            ('CV2', CountVectorizer(analyzer='char_wb',
                ngram_range=(1, 2)))
        ]
        combined = FeatureUnion(estimators, n_jobs=-1)

        for clf in clfs:
            pipelines.append(Pipeline([
                ('features', combined),
                ('TFIDF', TfidfTransformer(norm='l2')),
                ('clf', clf)
            ]))

    clfs_results = defaultdict(list)
    full_clfs_results = defaultdict(list)
    CV = StratifiedKFold(n_splits=10)
    for pipeline in pipelines:
        clf_name = type(pipeline.named_steps['clf']).
          __name__
        if clf_name != 'MultinomialNB':
            continue

        print(clf_name)
        for train_index, test_index in CV.split(X, y):
            pipeline.fit(X[train_index], y[train_index
              ])
            y_true, y_pred = y[test_index], pipeline.
              predict(X[test_index])
            print(classification_report(y_true, y_pred)
              )
            clfs_results[clf_name].append(
              precision_recall_fscore_support(y_true,
              y_pred, average='macro')[:3])
            full_clfs_results[clf_name].append(
              precision_recall_fscore_support(y_true,
              y_pred)[:3])

        print(clf_name)
        results = clfs_results[clf_name]
        print('\t'.join([
            'macro',
```

```
            str(round(sum([x[0] for x in results]) *
              100 / len(results), 2)),
            str(round(sum([x[1] for x in results]) *
              100 / len(results), 2)),
            str(round(sum([x[2] for x in results]) *
              100 / len(results), 2))
        ]))

        results = full_clfs_results[clf_name]
        print('\t'.join([
            'bad',
            str(round(sum([x[0][0] for x in results]) *
                100 / len(results), 2)),
            str(round(sum([x[1][0] for x in results]) *
                100 / len(results), 2)),
            str(round(sum([x[2][0] for x in results]) *
                100 / len(results), 2))
        ]))
        print('\t'.join([
            'good',
            str(round(sum([x[0][1] for x in results]) *
                100 / len(results), 2)),
            str(round(sum([x[1][1] for x in results]) *
                100 / len(results), 2)),
            str(round(sum([x[2][1] for x in results]) *
                100 / len(results), 2))
        ]))
```