Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

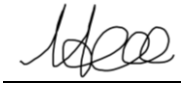
Институт: *ИДДО* Кафедра: *УИТ*

Направление подготовки: *27.03.04 Управление в технических системах*

ОТЧЕТ по практике

Наименование практики: Производственная практика: научно-исследовательская работа

СТУДЕНТ



/ Лебедкова М.А. /
(подпись) (*Фамилия и инициалы*)

Группа ИДз-31-19
(номер учебной группы)

ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ ПО ПРАКТИКЕ

(отлично, хорошо, удовлетворительно, неудовлетворительно, зачтено, не зачтено)

/ Бородкин А.А. /
(подпись) (*Фамилия и инициалы члена комиссии*)

/ /
(подпись) (*Фамилия и инициалы члена комиссии*)

Москва
2021

Оглавление

Введение

С каждым годом количество информации, представленной в электронном виде, стремительно растет. Это связано с тем, что в настоящее время доступность глобальной сети Интернет продолжает расти, происходит непрерывный процесс компьютеризации общества, а также большое количество текстов с бумажных носителей проходят процедуру перевода в электронный вид. Становится сложно анализировать большой объем неструктурированных текстовых данных одному человеку или даже группе экспертов. В связи с этим возникает потребность в ее автоматизированной обработке и анализе.

Классификация текстов – одна из важнейших задач обработки естественного языка. Это процесс классификации текстовых строк или документов по различным категориям в зависимости от содержания строк. Классификация текста имеет множество приложений, таких как определение настроений пользователей по твиту, классификация электронной почты как спама или, классификация сообщений в блогах по различным категориям, автоматическая маркировка запросов клиентов и т. д.

В данной работе рассматривается задача классификации отзывов к фильмам по трем классам: положительные, отрицательные и нейтральные. Это классический пример сентиментального анализа, когда чувства людей по отношению к тому или иному субъекту классифицируются по различным категориям. Отзывы взяты с портала Кинопоиск.

Теория

Задача классификации текстов на естественном языке (ЕЯ текстов) имеет практическое применение в следующих областях:

- сужение области поиска в поисковых системах;
- фильтрация спама; составление тематических каталогов;
- контекстная реклама;
- системы документооборота.

Востребованность в эффективном решении задачи автоматической классификации ЕЯ-текстов привела к бурному развитию новых методов и повышению эффективности уже существующих подходов.

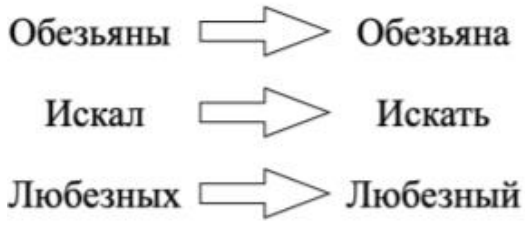
В настоящее время в решении задачи классификации текста можно выделить несколько основных составляющих:

1. Предобработка текста – процедура подготовки текста. Включает в себя методы токенизации отдельных слов/предложений, очистки текста от шума (стопслова, пунктуация и др.), морфологический анализ слов, исправление орфографии и др.
2. Векторизация текста – процедура представления текста в виде числовых векторов.
3. Применение одного из известных алгоритмов классификации. В анализе текстов для классификации очень часто из классических методов машинного обучения используется метод опорных векторов с линейным ядром (SVM).

Одна из самых важных задач из блока предобработки текстовой информации – это нормализация текста. Приведение слова к нормальной форме можно разделить на два совершенно разных подхода. Первый подход основан на необходимости объявить некий стандарт (лемму) для каждого слова и привести слово к этой лемме. Обычно леммами выступают слова в словарной форме. В общем случае, используются следующие правила:

- Существительное – именительный падеж, единственное число.
- Прилагательное – именительный падеж, единственное число, мужской род.
- Глагол – глагол в инфинитиве несовершенного вида.

Данный метод носит название лемматизация. Метод имеет наибольшее распространение в сфере обработки естественного языка.



. Пример работы алгоритма лемматизации

Полной противоположностью алгоритму лемматизации является метод стематизации, то есть нахождения основы для заданного слова. Сразу стоит отметить, что основа слова не всегда совпадает с морфологическим корнем. Главная идея заключается в том, что от слова отрезается по кусочку: и с конца, и сначала удаляются окончания, приставки, суффиксы, и в итоге остается основная часть слова. Пример применения стеммера Портера: слово «кошками» будет преобразовано в «кош». Основной проблемой методов нормализации текста является необходимость проведения больших лингвистических исследований при подключении нового языка.

Мешок слов (bag of words) – наиболее интуитивно понятный метод. Провести векторизацию текста – это значит представить его в виде следующей конструкции. Подсчитаем количество уникальных слов во всей обучающей выборке, и каждому слову присвоим один единственный индекс. Далее, для каждого документа i из обучающей выборки подсчитаем количество употреблений данного слова.

Мешок слов является хорошим базовым методом для решения задачи векторизации текста, однако алгоритм обладает рядом проблем:

1. Мешок слов никак не учитывает порядок слов. Например, если слова «гладить» и «собака» встречаются в выборке достаточно часто, это никак не учитывается алгоритмом.
2. В длинных примерах из обучающей выборки среднее количество употребления слов будет выше, чем в коротких.

Для решения первой проблемы можно воспользоваться расширением словаря N-граммами. N-грамма – это комбинация из n последовательных терминов для выделения из текста устойчивых словосочетаний. Сочетание из двух слов называются биграммой, из трех – триграмой, и т.д. В широком смысле N-граммой могут быть и определенные последовательности звуков, букв, слогов. Расширение признакового пространства N-граммами увеличивает скорость работы алгоритма, особенно для больших выборок, однако, в то же время, позволяет существенно увеличить точность работы алгоритма.

Вторая проблема находит решение в применении к алгоритму мешка слов статистической меры TF-IDF. TF-IDF – это мера для оценки важности конкретного признака в примере обучающей выборки. Вес признака в примере пропорционален частоте употребления данного признака в примере из обучающей выборки и обратно пропорционален частоте употребления признака во всех примерах обучающей выборки.

К преимуществам применения мешка слов и статистической меры TF-IDF можно отнести следующие факторы:

1. Позволяет строить векторное представление документов в условиях очень малого количества данных.
2. Алгоритм работает очень быстро за счет небольшого объема операций.
3. Простота реализации.

Алгоритм векторизации на основе мешка слов и TF-IDF в связке с некоторыми моделями машинного обучения показывает действительно хорошие результаты, но, к сожалению, только в условиях действительно репрезентативной выборки.

Такая связка имеет ряд минусов. Прежде всего, стоит отметить, что алгоритм никак не учитывает слова-синонимы, поскольку сама по себе частота слова не позволяет учитывать их. Вектор признаков может иметь очень большую размерность, что в свою очередь приведет к дополнительным затратам ресурсов для обучения классификатора, или для визуализации.

Задача классификации текстовой информации в современном мире может решаться поразному.

Для сравнения алгоритмов в данной работе было решено использовать алгоритмы машинного обучения из 3 различных классов.

- Ансамбль или комитет, который представляет собой обобщение большого количества базовых моделей, по отдельности дающие плохую точность, а в совокупности (ансамбле) хорошую. В данном классе был использован Случайный лес (Random forest).

- Линейные модели, делающие предположение о линейной сепарабельности классов, или, в случае восстановления уравнения регрессии, о зависимости между признаками, которая описывается линейно. В этом классе использовался метод опорных векторов с линейным ядром.

- Метод ближайших соседей – простой алгоритм из класса метрических алгоритмов.

Результаты работы данных алгоритмов и их сравнение приведено в практической части данной работы.

Практическая часть

1. *Предварительная обработка текста.*

Первым делом следует подключить все необходимые библиотеки.

```
import nltk
from nltk.stem.snowball import SnowballStemmer
from nltk.probability import FreqDist
from nltk.tokenize import RegexpTokenizer
from nltk import pos_tag
from collections import OrderedDict
import itertools
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn import svm
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.feature_extraction.text import TfidfVectorizer
import random
import numpy as np
import scipy.special
import os.path
import os
import matplotlib.pyplot as plt
from matplotlib.pylab import rc, plot
import seaborn as sns
from time import time
import pandas as pd
```

Теперь необходимо предварительно обработать и векторизовать наши данные. Предобработку будем выполнять, используя библиотеку NLTK. Для токенизации (разбиения текста на слова) будем использовать RegexpTokenizer из модуля nltk.tokenize. В него необходимо передать регулярное выражение. Далее слова необходимо привести к нижнему регистру и удалить служебные части речи. Для этого можно воспользоваться функцией pos_tag, которая присваивает каждому слову метку части речи (или ставит метку, что данная строка не является словом).

Также можно заметить, что в этой функции используется стэмминг, суть которого заключается в отбрасывании суффиксов и приставок у слов. Это позволяет сократить размерность признаков, так как слова с разными родами и падежами будут сокращены до одинаковых лексем. Для стемминга будем использовать SnowballStemmer из модуля nltk.stem.snowball. Существует более мощный аналог стэмминга – лемматизация, она позволяет восстановить начальную форму слова. Однако работает она медленнее стэмминга, и, помимо этого, в NLTK нет русского лемматизатора.

Функция обработки будет возвращать Pandas DataFrame, строки которого будут соответствовать файлам, для которых будут определены обработанный текст и тип отзыва.

```
def great_process(labels):
    data = {"File": [], "Words": [], "Label": []}
    tokenizer = RegexpTokenizer(r"\w+|[\W\s]+")
    stemmer = SnowballStemmer("russian")
    for label in labels:
        for c_dir, _, files in os.walk("D:\МЭИ\Практика\Отзывы к фильмам\{}".format(label)):
            for file in files:
                with open("D:\МЭИ\Практика\Отзывы к фильмам\{}\{}".format(label, file), encoding='utf-8') as f:
                    words = f.read()
                    bag_words = tokenizer.tokenize(words) # Токенезация
                    # Приведение к нижнему регистру
                    for i in bag_words:
                        i = i.lower()
                    # Присвоение частей речи
                    pos_words = pos_tag(bag_words, lang='rus')
                    # Очищение от служебных слов
                    cleaned_words = []
                    for i in pos_words:
                        if i[1] in ['S', 'A', 'V', 'ADV']:
                            cleaned_words.append(stemmer.stem(i[0]))
                text = " ".join(cleaned_words)
                data["File"].append(file[:-4] + '_' + label)
                data["Words"].append(text)
                data["Label"].append(label)
        print(label)
    return pd.DataFrame(data)
```

```
df = great_process(['neutral', 'bad', 'good'])
```

```
neutral
bad
good
```

```
df
```

	File	Words	Label
0	0_neutral	нача показател груб дерзк лома мейнстрим одаре...	neutral
1	1_neutral	откровен говор удивля феном режиссер тарантин ...	neutral
2	10_neutral	квентин тарантин скаж честн явля режиссер филь...	neutral
3	100_neutral	ход кинотеатр восторг чувств сторон действител...	neutral
4	1000_neutral	фанат очен долг ждал момент почт команд сбор л...	neutral
...	...	...	...
2995	995_good	моряк слишк долг плава успел позаб нрав дьявол...	good
2996	996_good	анонс выход экра част экшн франшиз форсаж был ...	good
2997	997_good	нача фильм шедевр фильм феном сдела кристофер ...	good
2998	998_good	призна честн посмотрел фильм вчер ран возможн ...	good
2999	999_good	смотрел фильм давн помн протяжен лент оторва б...	good

3000 rows × 3 columns

Делим данные на обучающую и тестовую выборку:

```
X_train, X_test, y_train, y_test = train_test_split(df['Words'],
                                                    df['Label'],
                                                    test_size=0.15,
                                                    random_state=8)
```

2. Векторизация текстовых документов.

Далее следует выполнить векторизацию, то есть представление наших текстовых данных в числовые, чтобы их смог обработать компьютер. Существует несколько способов векторизации. В данной работе будут использоваться TF-IDF векторы для представления отзывов.

TF-IDF значение увеличивается пропорционально количеству появлений слова в документе и смещается на количество документов в выборке, содержащих слово, что помогает отрегулировать тот факт, что некоторые слова появляются чаще. Данный метод также учитывает тот факт, что некоторые документы могут быть больше, чем другие, путем нормализации частот.

Выбор именно этого типа векторизации мотивирован следующими моментами:

- TF-IDF это простая модель, которая дает отличные результаты при анализе текстовой информации, как мы увидим позже;
- Это достаточно быстрый процесс, что важно при большом числе файлов;
- Мы можем контролировать процесс векторизации подбором параметров.

При создании объектов с помощью этого метода мы можем выбрать несколько параметров:

- Диапазон N-грамм (сочетания N соседних слов): мы можем рассматривать униграммы, биграммы, триграммы…
- Максимальная / минимальная частота в документах: при построении словаря мы можем игнорировать термины, у которых частота в документах строго выше / ниже заданного порога.
- Максимум параметров: мы можем выбрать N параметров (терминов), наиболее встречаемых в документах.

Логично ожидать, что биграммы и триграммы могут улучшить производительность модели. При подборе оптимальных параметров будут испытаны три варианта модели: с использованием униграмм; униграмм и биграмм; униграмм, биграмм и триграмм. Значение Minimum DF необходимо настроить, чтобы избавиться от чрезвычайно редких слов, которые не

появляются в более чем min_df документах (будем подбирать в пределах от 5 до 100 с шагом 5), а Maximum DF установим 100%, чтобы не игнорировать любые другие слова. Максимальное количество параметров необходимо настроить, чтобы избежать возможного переоснащения, часто возникающего из-за большого количества параметров по сравнению с количеством обучающих наблюдений (будем подбирать в пределах от 500 до 10000 с шагом 500).

```
# Функция для удобства подбора параметров векторизации
def Vector_Apply(ngram_range, min_df, max_features):
    tfidf = TfidfVectorizer(encoding='utf-8',
                            ngram_range=ngram_range,
                            stop_words=None,
                            lowercase=False,
                            max_df=1.,
                            min_df=min_df,
                            max_features=max_features,
                            norm='l2',
                            sublinear_tf=True)

    features_train = tfidf.fit_transform(X_train).toarray()
    labels_train = y_train

    features_test = tfidf.transform(X_test).toarray()
    labels_test = y_test
    return features_train, features_test, labels_train, labels_test
```

Для подбора параметров векторизации можно использовать модель Random Forest. Пока у нас нет возможности провести кросс валидацию и подбор оптимальных параметров модели. Возьмем глубину деревьев 15, а остальные параметры оставим по умолчанию.

```
ngram_range_list = [(1, 1), (1, 2), (1, 3)]
min_df_list = list(range(5, 101, 5))
max_features_list = list(range(500, 10001, 500))
res = {'ngram_range':[], 'min_df':[], 'max_features':[], 'score':[]}
for ngram_range in ngram_range_list:
    for min_df in min_df_list:
        for max_features in max_features_list:
            features_train, features_test, labels_train, labels_test = Vector_Apply(ngram_range, min_df, max_features)
            clf = RandomForestClassifier(max_depth=15)
            clf.fit(features_train, labels_train)
            predicted = clf.predict(features_test)
            score = accuracy_score(labels_test, predicted)
            res['ngram_range'].append(ngram_range)
            res['min_df'].append(min_df)
            res['max_features'].append(max_features)
            res['score'].append(score)
            print(max_features)
        print(min_df)
    print(ngram_range)
res_df = pd.DataFrame(res)
res_df
```

	ngram_range	min_df	max_features	score
0	(1, 1)	5	500	0.582222
1	(1, 1)	5	1000	0.575556
2	(1, 1)	5	1500	0.562222
3	(1, 1)	5	2000	0.582222
4	(1, 1)	5	2500	0.591111
...	...	...	...	...
1195	(1, 3)	100	8000	0.582222
1196	(1, 3)	100	8500	0.584444
1197	(1, 3)	100	9000	0.571111
1198	(1, 3)	100	9500	0.566667
1199	(1, 3)	100	10000	0.580000

1200 rows × 4 columns

```
max_score = res_df['score'].max()
res_df.query('score == @max_score')
```

	ngram_range	min_df	max_features	score
89	(1, 1)	25	5000	0.624444
348	(1, 1)	90	4500	0.624444

Как видно из проведенного исследования, максимальная точность была получена при двух разных сочетаниях параметров. Можно заметить, что использование биграмм и триграмм не сыграло решающей роли в качестве классификации. Так как у нас 2 возможных варианта параметров, рационально выбрать тот, в котором max_features меньше, так как это поможет сократить количество признаков и сэкономить время для обучения моделей.

```
params = res_df.query('(score == @max_score) & (max_features == 4500)')
ngram_range = params['ngram_range'].values[0]
min_df = params['min_df'].values[0]
max_features = params['max_features'].values[0]
features_train, features_test, labels_train, labels_test = Vector_Apply(ngram_range, min_df, max_features)
```

3. *Метод опорных векторов.*

Следующим шагом является применение методов машинного обучения. Начнем с метода опорных векторов. Так как в текстовых данных всегда большое число параметров (терминов), то лучше всего будет использовать линейную гиперплоскость для классификации. В следствие этого пропадает необходимость

в кросс-валидации для подбора параметров C и Гамма. Они будут равны 1 и 0 соответственно (значения по умолчанию).

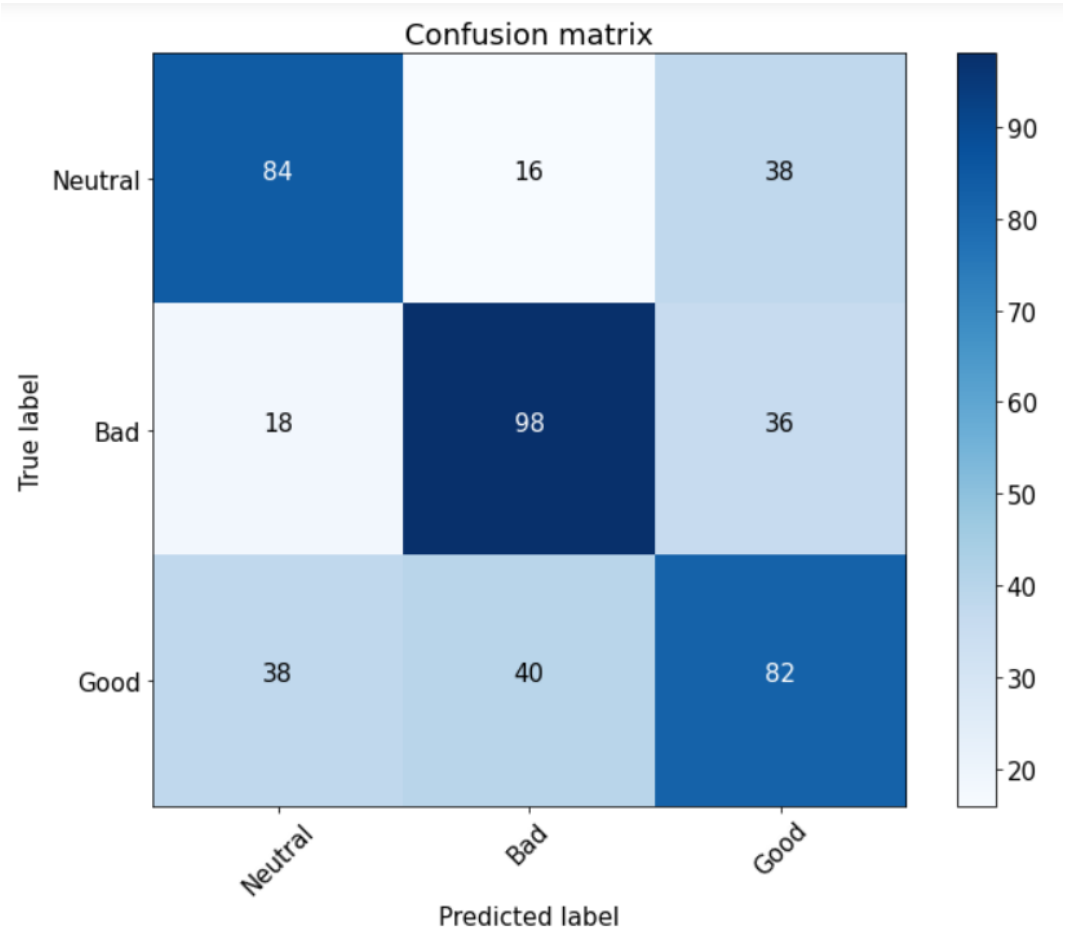
```
model = svm.SVC(kernel='linear')
model.fit(features_train, labels_train)
predicted_SVC = model.predict(features_test)
# Точность на контрольном датасете
score_test = accuracy_score(labels_test, predicted_SVC)
print(score_test)
# Классификационный отчет
report = classification_report(labels_test, predicted_SVC)
print(report)
```

```
0.5866666666666667
              precision    recall  f1-score   support

         bad       0.60      0.61      0.60       138
        good       0.64      0.64      0.64       152
     neutral       0.53      0.51      0.52       160

    accuracy                           0.59       450
   macro avg       0.59      0.59      0.59       450
weighted avg       0.59      0.59      0.59       450
```

Как можно заметить из классификационного отчета, точность определения нейтральных отзывов несколько ниже остальных. Вероятно, это связано с тем, что в нейтральных отзывах наблюдаются признаки как положительных, так и отрицательных типов. Визуализируем полученный результат в виде матрицы ошибок. Для этого напишем следующую функцию, которая представит данную матрицу в виде графика:

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

font = {'size' : 15}
plt.rc('font', **font)
```

```
cnf_matrix = confusion_matrix(labels_test, model.predict(features_test))
plt.figure(figsize=(10, 8))
plot_confusion_matrix(cnf_matrix, classes=['Neutral', 'Bad', 'Good'], title='Confusion matrix')
plt.savefig("conf_matrix.png")
plt.show()
```



Как видно из полученной матрицы, самое большое количество неправильно классифицированных отзывов именно среди нейтральных, что ещё раз подтверждает сделанный вывод.

4. *Метод Случайный Лес.*

Переходим к методу Случайный Лес, который использовался для подбора параметров векторизации. Попробуем оптимизировать его параметры. Мы будем использовать алгоритм RandomizedSearchCV. Он позволяет исследовать широкие диапазоны значений. Создается словарь params, в котором для каждого параметра задан диапазон его изменений. Затем создается объект RS с помощью функции RandomizedSearchCV(), передавая ей модель, params, число итераций и число кросс-валидаций, которые нужно выполнить.

Нас будут интересовать параметры:

- n_estimators — число «деревьев» в «случайном лесу».
- max_features — число признаков для выбора расщепления.
- max_depth — максимальная глубина деревьев.

```
n_estimators = list(range(100, 1001, 100))
max_features = ['sqrt', 'log2']
max_depth = list(range(1, 31))
param_dist = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth}
RS = RandomizedSearchCV(RandomForestClassifier(), 
                        param_dist,
                        n_iter = 100,
                        cv = 10,
                        verbose = 1,
                        n_jobs=-1,
                        random_state=0)
RS.fit(features_train, labels_train)
RS.best_params_
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   57.0s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  4.8min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 10.6min
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed: 16.8min
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed: 21.5min finished
```

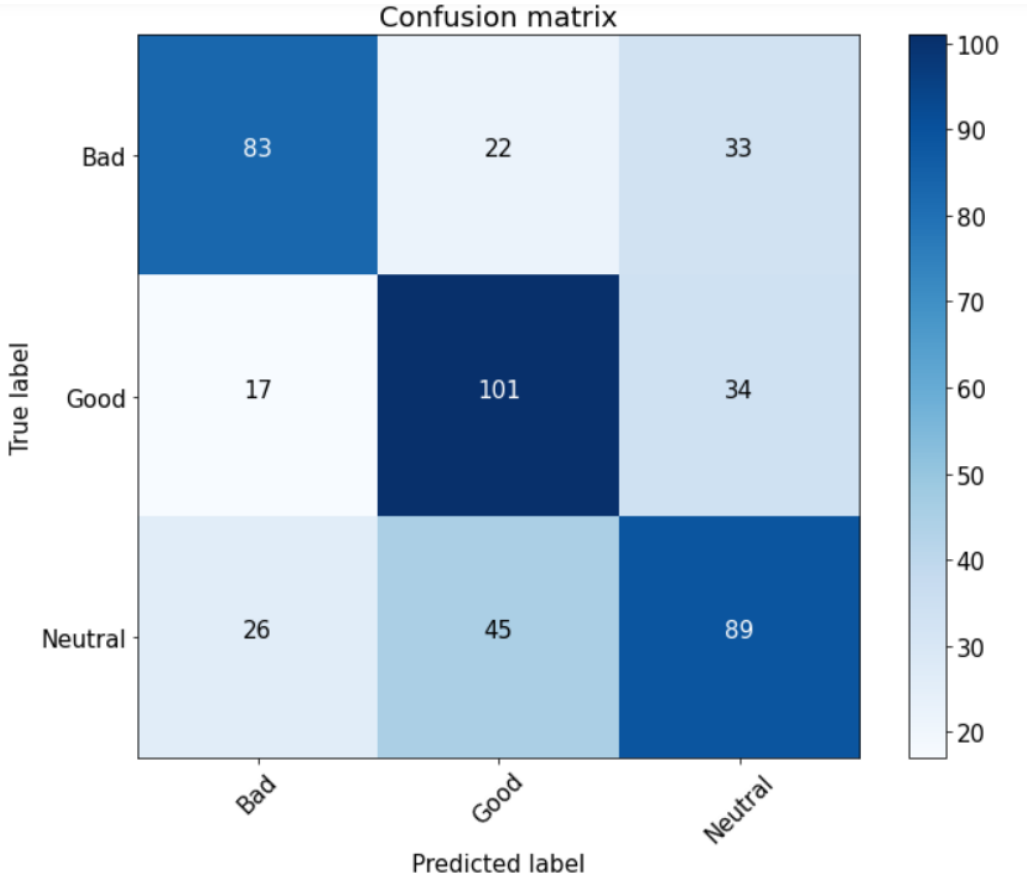
```
{'n_estimators': 900, 'max_features': 'log2', 'max_depth': 29}
```

Используем полученные параметры.

```
model = RandomForestClassifier(n_estimators=900, max_features='log2', max_depth=29)
model.fit(features_train, labels_train)
predicted_FC = model.predict(features_test)
# Точность на контрольном датасете
score_test = accuracy_score(labels_test, predicted_FC)
print(score_test)
# Классификационный отчет
report = classification_report(labels_test, predicted_FC)
print(report)
```

```
0.6066666666666667
              precision    recall  f1-score   support

         bad       0.66      0.60      0.63       138
        good       0.60      0.66      0.63       152
     neutral       0.57      0.56      0.56       160

    accuracy                           0.61       450
   macro avg       0.61      0.61      0.61       450
weighted avg       0.61      0.61      0.61       450
```

```
cnf_matrix = confusion_matrix(labels_test, model.predict(features_test))
plt.figure(figsize=(10, 8))
plot_confusion_matrix(cnf_matrix, classes=['Bad', 'Good', 'Neutral'], title='Confusion matrix')
plt.savefig("conf_matrix.png")
plt.show()
```



Данный метод показал более высокую точность, однако она по-прежнему остается небольшой. Причем больше всего ошибок первого и второго рода, как можно заметить по матрице ошибок, допускается в отношении нейтральных отзывов.

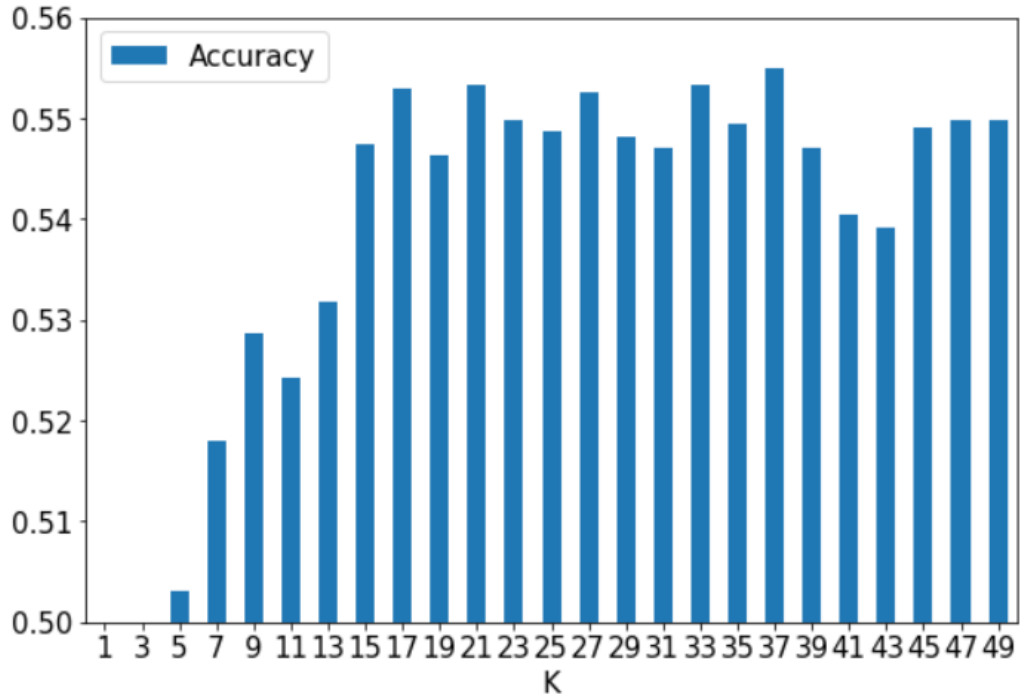
5. *Метод К ближайших соседей.*

Далее используем метод К ближайших соседей. Для него достаточно оптимизировать лишь параметр К. Сделать это можно с помощью k-кратной перекрестной проверки. Проведем 10-кратную перекрестную проверку, используя сгенерированный список нечетных К в диапазоне от 1 до 50.

```
neighbors = list(range(1, 50, 2))
cv_scores = [ ]
for K in neighbors:
    knn = KNeighborsClassifier(n_neighbors=K)
    scores = cross_val_score(knn, features_train, labels_train, cv=10, scoring="accuracy")
    cv_scores.append(scores.mean())
```

Оптимальное значение К можно получить путем построения показателя точности для разных значений К.

```
def plot_accuracy(scores):
    pd.DataFrame({"K":[i for i in range(1,50,2)], "Accuracy":scores}).set_index("K").plot.bar(figsize= (9,6),
                                                                                             ylim=(0.5,0.56),
                                                                                             rot=0)

    plt.show()
plot_accuracy(cv_scores)
```

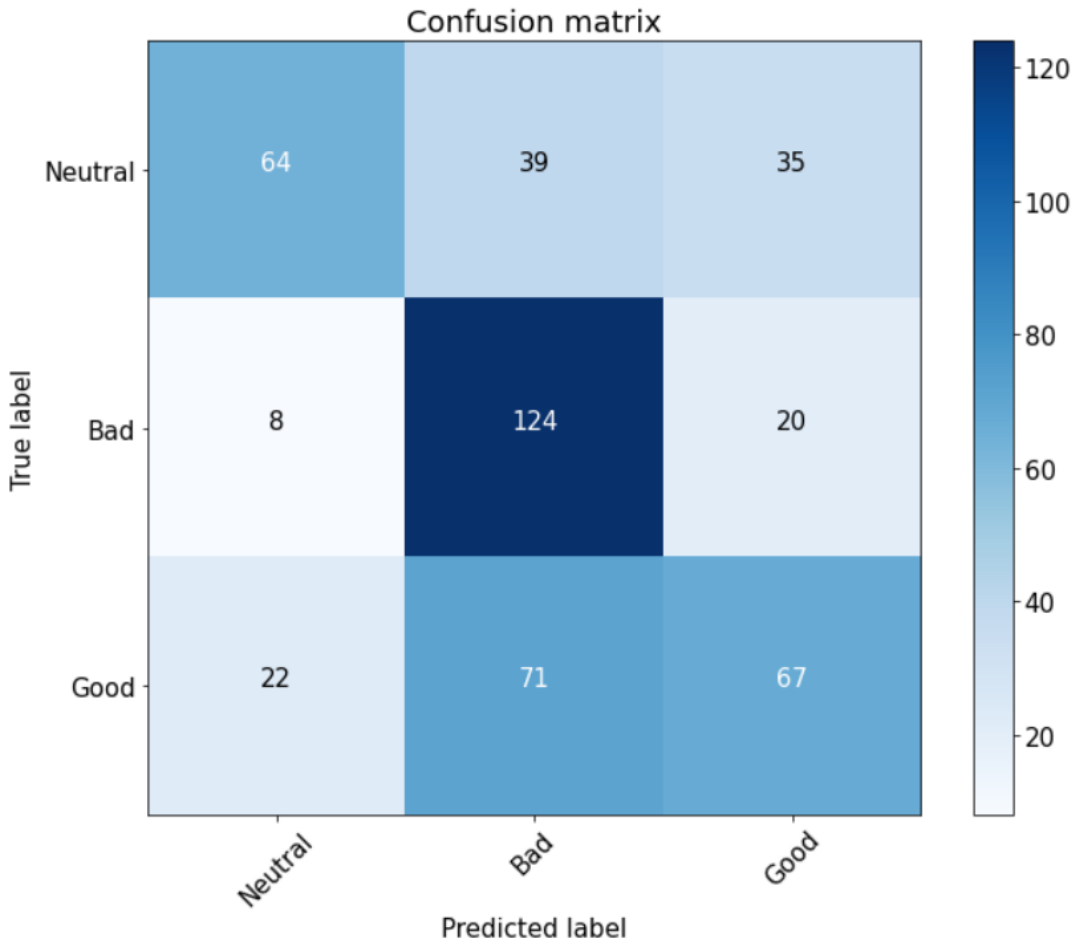


Используем полученное значение K = 37.

```
model = KNeighborsClassifier(n_neighbors=37)
model.fit(features_train, labels_train)
predicted_KN = model.predict(features_test)
# Точность на контрольном датасете
score_test = accuracy_score(labels_test, predicted_KN)
print(score_test)
report = classification_report(labels_test, predicted_KN)
print(report)
```

```
0.5666666666666667
              precision    recall  f1-score   support

         bad       0.68      0.46      0.55       138
        good       0.53      0.82      0.64       152
     neutral       0.55      0.42      0.48       160

    accuracy                           0.57       450
   macro avg       0.59      0.57      0.56       450
weighted avg       0.58      0.57      0.56       450
```

```
cnf_matrix = confusion_matrix(labels_test, model.predict(features_test))
plt.figure(figsize=(10, 8))
plot_confusion_matrix(cnf_matrix, classes=['Neutral', 'Bad', 'Good'], title='Confusion matrix')
plt.savefig("conf_matrix.png")
plt.show()
```



В данной модели очень хорошо определились плохие отзывы. А вот хорошие уступили даже нейтральным. Вообще, исходя из результатов моделей, кажется, что именно граница между хорошими и нейтральными отзывами размыта больше всего. Возможно, это связано с особенностями человеческого мышления.

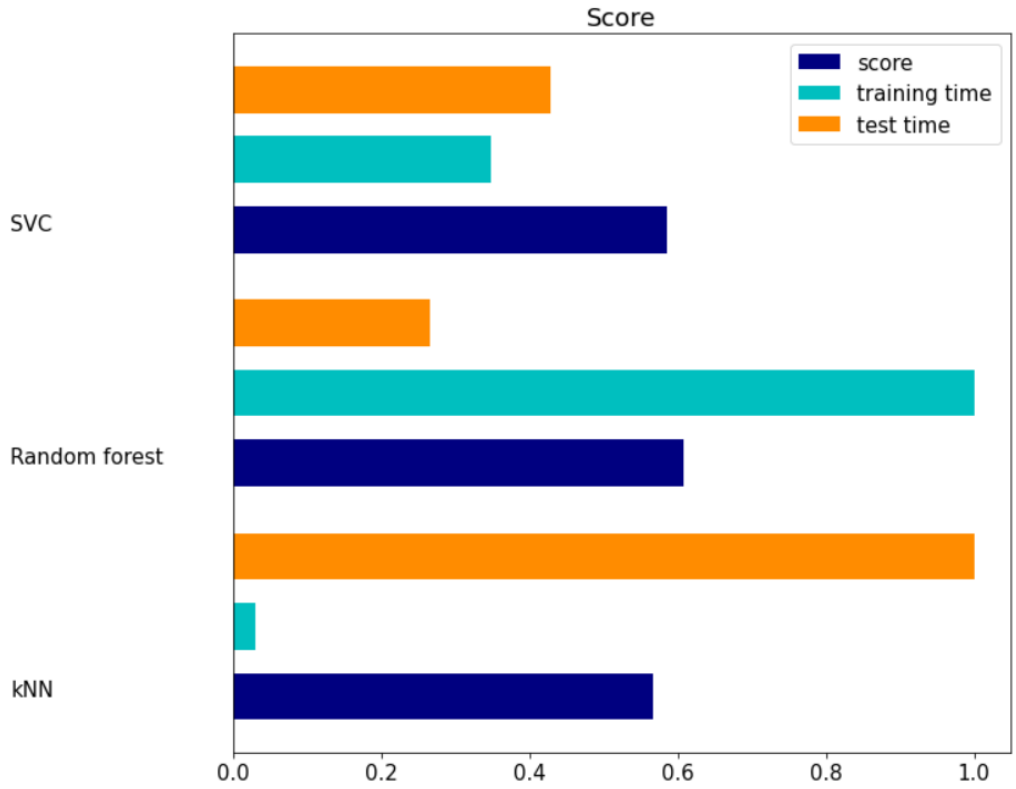
6. *Сравнительный анализ методов.*

Проведем краткий сравнительный анализ использованных методов. В качестве параметров для сравнения используем точность и время, затраченное на обучение и на анализ тестовой выборки. Напишем для визуализации сравнительной диаграммы следующую функцию.

```
def benchmark(clf, name):
    t0 = time()
    clf.fit(features_train, labels_train)
    train_time = time() - t0

    t0 = time()
    pred = clf.predict(features_test)
    test_time = time() - t0

    score = accuracy_score(labels_test, pred)

    return name, score, train_time, test_time
```

```
results = []
for clf, name in ((KNeighborsClassifier(n_neighbors=37), "kNN"),
                  (RandomForestClassifier(n_estimators=900, max_features='log2', max_depth=29), "Random forest"),
                  (svm.SVC(kernel='linear'), "SVC")):
    results.append(benchmark(clf, name))
```

```
indices = np.arange(len(results))

results = [[x[i] for x in results] for i in range(4)]

clf_names, score, training_time, test_time = results
training_time = np.array(training_time) / np.max(training_time)
test_time = np.array(test_time) / np.max(test_time)

plt.figure(figsize=(12, 8))
plt.title("Score")
plt.barh(indices, score, .2, label="score", color='navy')
plt.barh(indices + .3, training_time, .2, label="training time",
         color='c')
plt.barh(indices + .6, test_time, .2, label="test time", color='darkorange')
plt.yticks(())
plt.legend(loc='best')
plt.subplots_adjust(left=.25)
plt.subplots_adjust(top=.95)
plt.subplots_adjust(bottom=.05)

for i, c in zip(indices, clf_names):
    plt.text(-.3, i, c)

plt.show()
```



Исходя из полученной диаграммы можно сделать вывод, что самым точным оказался метод Случайный лес. Однако, он же и самый затратный по времени обучения. Самым оптимальным сочетанием точности и временных затрат обладает метод опорных векторов. А вот метод К ближайших соседей явно не подходит для анализа текстовых данных. Он обладает минимальной точностью при максимальном времени анализа тестовых данных. Скорее всего, он чувствителен к большому числу признаков, которого при анализе текстовой информации не избежать.

Заключение

Исходя из результатов проведенных исследований, можно сделать вывод, что самым лучшим образом себя показал алгоритм опорных векторов. У него наблюдается лучшее соотношение временных затрат и результатов работы. В изученной литературе данный метод также выделялся, как один из лучших среди классических методов машинного обучения при анализе текстовой информации. Однако, стоит отметить, что добиться достаточно высокой точности классификации ни в одном из методов так и не удалось. Это может быть связано с тем, что нейтральный класс отзывов не имеет четкой границы с положительным и отрицательным классами. Возможно, выделение только двух классов улучшило бы результаты работы моделей.

Список используемой литературы

1. Бурлаева Е. И. Обзор методов классификации текстовых документов на основе подхода машинного обучения. "Программная инженерия" Том 8, № 7, 2017. С. 328-336.
2. Злочевский Ю.И., Лебеденко А.В. Разработка системы семантического анализа данных в социальных сетях на основе машинных методов обучения с целью выявления таргетированной информации. «Морская стратегия и политика России в контексте обеспечения национальной безопасности и устойчивого развития в XXI веке». Сборник научных трудов. Севастополь, 2019. С. 252-256.
3. Шимановичс Кирс, Скородумова Е.А. Математическое моделирование системы интеллектуального анализа текста с целью построения QA системы. Телекоммуникации и информационные технологии. 2020. Т. 7. № 2. С. 131-137.