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 ИУ5 «Системы обработки информации и управления»

Отчёт по лабораторной работе
по дисциплине «Методы Машинного Обучения»

Выполнил:
студент группы № ИУ5-21М
Кучеренко Михаил Александрович
_____, _____

Проверил:
к.т.н., доц., Ю.Е. Гапанюк
_____, _____

2021 г.

Лабораторная работа №6

Классификация текста.

Набор данных - 20 newsgroups text dataset

Классы: 20
Выборка: 18846

Install

```
pip3 install gensim
pip3 install spacy
python3 -m spacy download en_core_web_sm
```

Задание:

Для произвольного набора данных, предназначенного для классификации текстов, решите задачу классификации текста двумя способами:

- Способ 1. На основе CountVectorizer или TfidfVectorizer.
- Способ 2. На основе моделей word2vec или Glove или fastText.
- Сравните качество полученных моделей.

Для поиска наборов данных в поисковой системе можно использовать ключевые слова "datasets for text classification".

```
import nltk
import spacy
import numpy as np
import sklearn
from sklearn.datasets import fetch_20newsgroups
from nltk import tokenize
import re
import pandas as pd
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /Users/snipghost/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
True
```

```
categories = ["sci.crypt", "sci.electronics", "talk.religion.misc"]
newsgroups_train = fetch_20newsgroups(subset='train',
categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)
```

```
unique, frequency = np.unique(newsgroups_train.target, return_counts=True)
```

```
for l, f in zip(unique, frequency):
    print(f'value: {l}, count: {f}')
```

```
value: 0, count: 595
value: 1, count: 591
value: 2, count: 377
```

```
from spacy.lang.en import English
import spacy
from nltk.corpus import stopwords

nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])
nltk.download('stopwords')
stopwords_eng = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/snipghost/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
def prepare(t):
    # t = ' '.join([i.strip().lower() for i in t.split(' ')])
    t = re.sub(r'[^a-zA-Z0-9 \n]', '', t)
    t = re.sub('\s+', ' ', t)
    t = ' '.join([token.lemma_.lower() for token in nlp(t) if token not in stopwords_eng])
    return t

texts = newsgroups_train.data
texts_array = []

for text in texts:
    prepared_text = prepare(text)
    texts_array.append(prepared_text)
```

```
print(len(texts_array), texts_array[0])
```

```
1563 from mhaldlynxdacnortheasternedu mark hald subject re dayton hamfest organization northeastern university boston ma 02115 usa distribution usa lines 13 i book a hotel red roof inn last week in cincinnati blue ash which be at the northern tip of the metro cincy area i choose it for a few reason 1 all hotel in and near dayton be book solid 2 this hotel be only cost 28night 3 it be one of about 4 room leave on the night i reserve 4 cincinnati probably have more to to at night than dayton i intend to hit the riverboat entertainment at dusk if anyone have other suggestion for nightlife please let i know of other hot spot thanks mark
```

```
test_texts = newsgroups_test.data
test_texts_arr = []

for text in test_texts:
    prepared_text = prepare(text)
    test_texts_arr.append(prepared_text)
```

```
print(len(test_texts_arr), test_texts_arr[0])
```

```
1040 from markkcypresswestsuncom mark kampe subject re cybele and
transgendersexualism organization sunsoft south lines 29 distribution world
replyto markkcypresswestsuncom nntppostinghost sagredo in article
260493115730ravenaimsuncedu fhuntmeduncedu freb hunt write be there some
relation between the name cybele and the phenemenon of the sibyl your
paragraph above seem to indicate there might be the oed give the etymology
of sibyl as come from the ancient greek sigma iota beta upsilon lambda
lambda alpha s i b ih l l a which be claim to come from the doric sigma
iota omicron beta upsilon lambda lambda alpha s i o b ih l l a which if i
read it properly in turn come from the attican athenian theta epsilon
omicron beta omicron upsilon lambda eta th eh o b o ih l ae i do nt know
much about attis but it would nt surprise i to learn that this god be tie
to the athenian capital alpha tau tau iota kappa upsilon sigma a t t i k u
s the oed do not list any etymology for cybele since that be a propper noun
but i suggest that the greek spelling of that word would be much close to
the anticedant of sibyl than the two word be now perhaps cybele be a french
or latin spelling
```

Способ 1.

TF-IDF + CountVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
```

```
tfidf_vectorizer = TfidfVectorizer()

train_feature_matrix_tfidf = tfidf_vectorizer.fit_transform(texts_array)
test_feature_matrix__tfidf = tfidf_vectorizer.transform(test_texts_arr)
```

```
count_vectorizer = CountVectorizer()

train_feature_matrix_count = count_vectorizer.fit_transform(texts_array)
test_feature_matrix_count = count_vectorizer.transform(test_texts_arr)
```

```
target_values_train = newsgroups_train.target
target_values_test = newsgroups_test.target
```

```
# sklearn.metrics.SCORERS.keys()
```

```
knn_tfidf = KNeighborsClassifier()
parameters = {'n_neighbors': [2, 3, 5, 7, 9, 11]}

knn_tfidf_grid = GridSearchCV(knn_tfidf, parameters,
scoring='balanced_accuracy', verbose=4, cv=5)
```

```
knn_tfidf_grid.fit(train_feature_matrix_count, target_values_train)

print('best param of n_neighbors',
knn_count_grid.best_params_['n_neighbors'])
best_knn_tfidf =
KNeighborsClassifier(n_neighbors=knn_count_grid.best_params_['n_neighbors']
)

best_knn_tfidf.fit(train_feature_matrix_tfidf, target_values_train)
best_pred_knn = best_knn_tfidf.predict(test_feature_matrix__tfidf)

print(classification_report(target_values_test, best_pred_knn))
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV] n_neighbors=2 ...................................................
[CV] ....................... n_neighbors=2, score=0.714, total=   0.1s
[CV] n_neighbors=2 ...................................................
[CV] ....................... n_neighbors=2, score=0.715, total=   0.1s
[CV] n_neighbors=2 ...................................................
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:
0.0s
```

```
[CV] ....................... n_neighbors=2, score=0.722, total=   0.1s
[CV] n_neighbors=2 ...................................................
[CV] ....................... n_neighbors=2, score=0.703, total=   0.1s
[CV] n_neighbors=2 ...................................................
[CV] ....................... n_neighbors=2, score=0.709, total=   0.1s
[CV] n_neighbors=3 ...................................................
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.2s remaining:
0.0s
```

```
[CV] ....................... n_neighbors=3, score=0.764, total=   0.1s
[CV] n_neighbors=3 ...................................................
```

```
[CV] ..... n_neighbors=3, score=0.759, total=   0.1s
[CV] n_neighbors=3 ....................................................
[CV] ..... n_neighbors=3, score=0.727, total=   0.1s
[CV] n_neighbors=3 ....................................................
[CV] ..... n_neighbors=3, score=0.710, total=   0.1s
[CV] n_neighbors=3 ....................................................
[CV] ..... n_neighbors=3, score=0.752, total=   0.1s
[CV] n_neighbors=5 ....................................................
[CV] ..... n_neighbors=5, score=0.758, total=   0.1s
[CV] n_neighbors=5 ....................................................
[CV] ..... n_neighbors=5, score=0.759, total=   0.1s
[CV] n_neighbors=5 ....................................................
[CV] ..... n_neighbors=5, score=0.721, total=   0.2s
[CV] n_neighbors=5 ....................................................
[CV] ..... n_neighbors=5, score=0.736, total=   0.1s
[CV] n_neighbors=5 ....................................................
[CV] ..... n_neighbors=5, score=0.723, total=   0.2s
[CV] n_neighbors=7 ....................................................
[CV] ..... n_neighbors=7, score=0.711, total=   0.1s
[CV] n_neighbors=7 ....................................................
[CV] ..... n_neighbors=7, score=0.715, total=   0.1s
[CV] n_neighbors=7 ....................................................
[CV] ..... n_neighbors=7, score=0.730, total=   0.1s
[CV] n_neighbors=7 ....................................................
[CV] ..... n_neighbors=7, score=0.706, total=   0.1s
[CV] n_neighbors=7 ....................................................
[CV] ..... n_neighbors=7, score=0.724, total=   0.1s
[CV] n_neighbors=9 ....................................................
[CV] ..... n_neighbors=9, score=0.702, total=   0.1s
[CV] n_neighbors=9 ....................................................
[CV] ..... n_neighbors=9, score=0.716, total=   0.1s
[CV] n_neighbors=9 ....................................................
[CV] ..... n_neighbors=9, score=0.720, total=   0.1s
[CV] n_neighbors=9 ....................................................
[CV] ..... n_neighbors=9, score=0.680, total=   0.1s
[CV] n_neighbors=9 ....................................................
[CV] ..... n_neighbors=9, score=0.716, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.651, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.701, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.706, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.673, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.713, total=   0.1s
best param of n_neighbors 3
```

```
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    3.1s finished
```

```
              precision    recall  f1-score   support

           0       0.73      0.96      0.83       396
           1       0.97      0.68      0.80       393
           2       0.89      0.87      0.88       251

    accuracy                           0.83      1040
   macro avg       0.86      0.84      0.84      1040
weighted avg       0.86      0.83      0.83      1040
```

```
knn_count = KNeighborsClassifier()
parameters = {'n_neighbors': [2, 3, 5, 7, 9, 11]}

knn_count_grid = GridSearchCV(knn_count, parameters,
scoring='balanced_accuracy', verbose=4, cv=5)
```

```
knn_count_grid.fit(train_feature_matrix_count, target_values_train)

print('best param of n_neighbors',
knn_count_grid.best_params_['n_neighbors'])
best_knn_count =
KNeighborsClassifier(n_neighbors=knn_count_grid.best_params_['n_neighbors']
)
print(best_knn_count)
best_knn_count.fit(train_feature_matrix_count, target_values_train)
best_knn_pred_count = best_knn_count.predict(test_feature_matrix_count)

print(classification_report(target_values_test, best_knn_pred_count))
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV] n_neighbors=2 ...................................................
[CV] ....................... n_neighbors=2, score=0.714, total=   0.1s
[CV] n_neighbors=2 ...................................................
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:
0.0s
```

```
[CV] ..... n_neighbors=2, score=0.715, total=   0.1s
[CV] n_neighbors=2 .....................................................
[CV] ..... n_neighbors=2, score=0.722, total=   0.1s
[CV] n_neighbors=2 .....................................................
[CV] ..... n_neighbors=2, score=0.703, total=   0.1s
[CV] n_neighbors=2 .....................................................
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.4s remaining:
0.0s
```

```
[CV] ..... n_neighbors=2, score=0.709, total=   0.1s
[CV] n_neighbors=3 .....................................................
[CV] ..... n_neighbors=3, score=0.764, total=   0.1s
[CV] n_neighbors=3 .....................................................
[CV] ..... n_neighbors=3, score=0.759, total=   0.1s
[CV] n_neighbors=3 .....................................................
[CV] ..... n_neighbors=3, score=0.727, total=   0.2s
[CV] n_neighbors=3 .....................................................
[CV] ..... n_neighbors=3, score=0.710, total=   0.1s
[CV] n_neighbors=3 .....................................................
[CV] ..... n_neighbors=3, score=0.752, total=   0.1s
[CV] n_neighbors=5 .....................................................
[CV] ..... n_neighbors=5, score=0.758, total=   0.1s
[CV] n_neighbors=5 .....................................................
[CV] ..... n_neighbors=5, score=0.759, total=   0.1s
[CV] n_neighbors=5 .....................................................
[CV] ..... n_neighbors=5, score=0.721, total=   0.1s
[CV] n_neighbors=5 .....................................................
[CV] ..... n_neighbors=5, score=0.736, total=   0.1s
[CV] n_neighbors=5 .....................................................
[CV] ..... n_neighbors=5, score=0.723, total=   0.1s
[CV] n_neighbors=7 .....................................................
[CV] ..... n_neighbors=7, score=0.711, total=   0.1s
[CV] n_neighbors=7 .....................................................
[CV] ..... n_neighbors=7, score=0.715, total=   0.1s
[CV] n_neighbors=7 .....................................................
[CV] ..... n_neighbors=7, score=0.730, total=   0.1s
[CV] n_neighbors=7 .....................................................
[CV] ..... n_neighbors=7, score=0.706, total=   0.2s
[CV] n_neighbors=7 .....................................................
[CV] ..... n_neighbors=7, score=0.724, total=   0.1s
[CV] n_neighbors=9 .....................................................
[CV] ..... n_neighbors=9, score=0.702, total=   0.1s
[CV] n_neighbors=9 .....................................................
[CV] ..... n_neighbors=9, score=0.716, total=   0.1s
[CV] n_neighbors=9 .....................................................
```

```
[CV] ..... n_neighbors=9, score=0.720, total=   0.1s
[CV] n_neighbors=9 ....................................................
[CV] ..... n_neighbors=9, score=0.680, total=   0.2s
[CV] n_neighbors=9 ....................................................
[CV] ..... n_neighbors=9, score=0.716, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.651, total=   0.2s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.701, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.706, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.673, total=   0.1s
[CV] n_neighbors=11 ...................................................
[CV] ..... n_neighbors=11, score=0.713, total=   0.1s
best param of n_neighbors 3
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

```
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    3.4s finished
```

	precision	recall	f1-score	support
0	0.60	0.74	0.66	396
1	0.67	0.61	0.64	393
2	0.73	0.57	0.64	251
accuracy			0.65	1040
macro avg	0.67	0.64	0.65	1040
weighted avg	0.66	0.65	0.65	1040

Способ 2.

```
import tqdm
from gensim.models import Word2Vec
import gensim.downloader
```

```
/Users/snipghost/opt/anaconda3/lib/python3.7/site-packages/gensim/similarities/__init__.py:15: UserWarning: The gensim.similarities.levenshtein submodule is disabled, because the optional Levenshtein package <https://pypi.org/project/python-Levenshtein/> is unavailable. Install Levenhstein (e.g. `pip install python-Levenshtein`) to suppress this warning.
  warnings.warn(msg)
```

```
gensim.downloader.info()
# glove_vectors = gensim.downloader.load('glove-twitter-25')
glove_vectors = gensim.downloader.load('glove-wiki-gigaword-50')
```

```
[==================================================] 100.0% 66.0/66.0MB downloaded
```

```
class GloveTokenizer:
    def __init__(self, glove_tokenizer):
        self.glove = glove_tokenizer
        self.token_length = 800
        self.embedding_size = 50

    def __getitem__(self, word):
        try:
            vector = glove_vectors.get_vector(word).reshape(1, self.embedding_size)
        except KeyError as e:
            vector = np.zeros((1, self.embedding_size))
        return vector


    def __padd(self, sentence):
        padded_sentence = np.zeros((self.token_length, self.embedding_size))
        for i, token in enumerate(sentence):
            padded_sentence[i] = token
        return padded_sentence

    def tokenize(self, sentence):
        encoded_sentence = []
        sentence = sentence.strip(' ').split(' ')
        for i in sentence:
            token = self.__getitem__(i)
            encoded_sentence.append(token)
```

```
        return np.array(self.__padd(encoded_sentence), dtype=np.float16)
```

```
tokenizer = GloveTokenizer(glove_vectors)
```

```
def prepare(t):
    # t = ' '.join([i.strip().lower() for i in t.split(' ')])
    t = re.sub(r'[^a-zA-Z0-9 ]', '', t)
    t = re.sub('\s+', ' ', t)
    lemmas = [token.lemma_.lower() for token in nlp(t) if token not in stopwords_eng]
    t = ' '.join(lemmas)
    vectors = tokenizer.tokenize(t)
    return vectors, len(lemmas)

vectors_array_train = []
labels_train = []

for enum, text, label in zip(range(len(newsgroups_train.data)), newsgroups_train.data, newsgroups_train.target):
    try:
        vector, length = prepare(text)
        # print(vector, vector.shape)
        vectors_array_train.append(vector)
        labels_train.append(label)
    except IndexError as e:
        print(enum, e)
        continue


vectors_array_train = np.array(vectors_array_train)
print(vectors_array_train.shape)
train_data = vectors_array_train.reshape((-1, vectors_array_train.shape[1]*vectors_array_train.shape[2]))
train_data.shape
```

```
18 index 800 is out of bounds for axis 0 with size 800
47 index 800 is out of bounds for axis 0 with size 800
186 index 800 is out of bounds for axis 0 with size 800
204 index 800 is out of bounds for axis 0 with size 800
211 index 800 is out of bounds for axis 0 with size 800
214 index 800 is out of bounds for axis 0 with size 800
261 index 800 is out of bounds for axis 0 with size 800
279 index 800 is out of bounds for axis 0 with size 800
313 index 800 is out of bounds for axis 0 with size 800
318 index 800 is out of bounds for axis 0 with size 800
330 index 800 is out of bounds for axis 0 with size 800
334 index 800 is out of bounds for axis 0 with size 800
```

```
355 index 800 is out of bounds for axis 0 with size 800
372 index 800 is out of bounds for axis 0 with size 800
389 index 800 is out of bounds for axis 0 with size 800
405 index 800 is out of bounds for axis 0 with size 800
418 index 800 is out of bounds for axis 0 with size 800
420 index 800 is out of bounds for axis 0 with size 800
425 index 800 is out of bounds for axis 0 with size 800
442 index 800 is out of bounds for axis 0 with size 800
446 index 800 is out of bounds for axis 0 with size 800
457 index 800 is out of bounds for axis 0 with size 800
463 index 800 is out of bounds for axis 0 with size 800
485 index 800 is out of bounds for axis 0 with size 800
549 index 800 is out of bounds for axis 0 with size 800
585 index 800 is out of bounds for axis 0 with size 800
637 index 800 is out of bounds for axis 0 with size 800
640 index 800 is out of bounds for axis 0 with size 800
654 index 800 is out of bounds for axis 0 with size 800
655 index 800 is out of bounds for axis 0 with size 800
670 index 800 is out of bounds for axis 0 with size 800
697 index 800 is out of bounds for axis 0 with size 800
703 index 800 is out of bounds for axis 0 with size 800
747 index 800 is out of bounds for axis 0 with size 800
755 index 800 is out of bounds for axis 0 with size 800
756 index 800 is out of bounds for axis 0 with size 800
758 index 800 is out of bounds for axis 0 with size 800
820 index 800 is out of bounds for axis 0 with size 800
834 index 800 is out of bounds for axis 0 with size 800
877 index 800 is out of bounds for axis 0 with size 800
892 index 800 is out of bounds for axis 0 with size 800
906 index 800 is out of bounds for axis 0 with size 800
949 index 800 is out of bounds for axis 0 with size 800
955 index 800 is out of bounds for axis 0 with size 800
959 index 800 is out of bounds for axis 0 with size 800
979 index 800 is out of bounds for axis 0 with size 800
985 index 800 is out of bounds for axis 0 with size 800
1041 index 800 is out of bounds for axis 0 with size 800
1115 index 800 is out of bounds for axis 0 with size 800
1120 index 800 is out of bounds for axis 0 with size 800
1122 index 800 is out of bounds for axis 0 with size 800
1124 index 800 is out of bounds for axis 0 with size 800
1125 index 800 is out of bounds for axis 0 with size 800
1138 index 800 is out of bounds for axis 0 with size 800
1140 index 800 is out of bounds for axis 0 with size 800
1150 index 800 is out of bounds for axis 0 with size 800
1153 index 800 is out of bounds for axis 0 with size 800
1165 index 800 is out of bounds for axis 0 with size 800
1185 index 800 is out of bounds for axis 0 with size 800
1186 index 800 is out of bounds for axis 0 with size 800
1191 index 800 is out of bounds for axis 0 with size 800
```

```
1201 index 800 is out of bounds for axis 0 with size 800
1209 index 800 is out of bounds for axis 0 with size 800
1260 index 800 is out of bounds for axis 0 with size 800
1262 index 800 is out of bounds for axis 0 with size 800
1295 index 800 is out of bounds for axis 0 with size 800
1321 index 800 is out of bounds for axis 0 with size 800
1339 index 800 is out of bounds for axis 0 with size 800
1426 index 800 is out of bounds for axis 0 with size 800
1489 index 800 is out of bounds for axis 0 with size 800
1502 index 800 is out of bounds for axis 0 with size 800
1545 index 800 is out of bounds for axis 0 with size 800
1551 index 800 is out of bounds for axis 0 with size 800
1561 index 800 is out of bounds for axis 0 with size 800
(1489, 800, 50)
```

```
(1489, 40000)
```

```
vectors_array_test = []
labels_test= []

for enum, text, label in zip(range(len(newsgroups_test.data)),
newsgroups_test.data, newsgroups_test.target):
    try:
        vector, length = prepare(text)
        vectors_array_test.append(vector)
        labels_test.append(label)
    except IndexError as e:
        print(enum, e)
        continue
```

```
46 index 800 is out of bounds for axis 0 with size 800
56 index 800 is out of bounds for axis 0 with size 800
69 index 800 is out of bounds for axis 0 with size 800
106 index 800 is out of bounds for axis 0 with size 800
122 index 800 is out of bounds for axis 0 with size 800
184 index 800 is out of bounds for axis 0 with size 800
282 index 800 is out of bounds for axis 0 with size 800
286 index 800 is out of bounds for axis 0 with size 800
341 index 800 is out of bounds for axis 0 with size 800
376 index 800 is out of bounds for axis 0 with size 800
402 index 800 is out of bounds for axis 0 with size 800
439 index 800 is out of bounds for axis 0 with size 800
```

```
455 index 800 is out of bounds for axis 0 with size 800
459 index 800 is out of bounds for axis 0 with size 800
505 index 800 is out of bounds for axis 0 with size 800
613 index 800 is out of bounds for axis 0 with size 800
651 index 800 is out of bounds for axis 0 with size 800
668 index 800 is out of bounds for axis 0 with size 800
794 index 800 is out of bounds for axis 0 with size 800
808 index 800 is out of bounds for axis 0 with size 800
884 index 800 is out of bounds for axis 0 with size 800
997 index 800 is out of bounds for axis 0 with size 800
999 index 800 is out of bounds for axis 0 with size 800
1027 index 800 is out of bounds for axis 0 with size 800
1034 index 800 is out of bounds for axis 0 with size 800
```

```
vectors_array_test = np.array(vectors_array_test)
test_data = vectors_array_test.reshape((-1,
vectors_array_test.shape[1]*vectors_array_test.shape[2]))
test_data.shape
```

```
(1015, 40000)
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report


glove_knn_clf = KNeighborsClassifier()
parameters = {'n_neighbors': [2, 3, 5, 7, 9]}

glove_clf_grid = GridSearchCV(glove_knn_clf, parameters, verbose=4, cv=3,
                              scoring='balanced_accuracy', n_jobs=-1)

glove_clf_grid.fit(train_data, labels_train)

print('best param of n_neighbors',
glove_clf_grid.best_params_['n_neighbors'])
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  12 out of  15 | elapsed:  4.7min remaining:  1.2min
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:  5.7min finished
```

```
best param of n_neighbors 7
```

```
best_glove_knn = KNeighborsClassifier(n_neighbors=glove_clf_grid.best_params_['n_neighbors'])
best_glove_knn.fit(train_data, labels_train)
best_pred_glove_knn = best_glove_knn.predict(test_data[:800])

print(classification_report(labels_test[:800], best_pred_glove_knn))
```

```
              precision    recall  f1-score   support

           0       0.49      0.54      0.51       316
           1       0.53      0.50      0.51       305
           2       0.48      0.42      0.45       179

    accuracy                           0.50       800
   macro avg       0.50      0.49      0.49       800
weighted avg       0.50      0.50      0.50       800
```

```

```