Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Лабораторная работа №6**

**По курсу «Методы машинного обучения»**

**ИСПОЛНИТЕЛЬ:**

Лю Цзыцзянь

Группа ИУ5И-21М

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ПРОВЕРИЛ:**

Гапанюк Ю.Е.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оглавление

[Лабораторная работа №6 3](#_Toc200477305)

[Импортируйте необходимые библиотеки и отобразите данные 4](#_Toc200477306)

[Векторизация признаков на основе CountVectorizer 10](#_Toc200477307)

[Разделим на обучающую и тестовую выборки 13](#_Toc200477308)

[Классификация текста на основе моделей word2vec 14](#_Toc200477309)

# Лабораторная работа №6

**Классификация текста.**

**Цель лабораторной работы: изучение методов классификации текстов.**

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".

# Импортируйте необходимые библиотеки и отобразите данные

**import** numpy **as** np

**import** pandas **as** pd

**from** typing **import** Dict, Tuple

**from** scipy **import** stats

**from** IPython.display **import** Image

**from** sklearn.datasets **import** load\_iris, load\_boston

**from** sklearn.feature\_extraction.text **import** CountVectorizer, TfidfVectorizer

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.neighbors **import** KNeighborsRegressor, KNeighborsClassifier

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.model\_selection **import** GridSearchCV, RandomizedSearchCV

**from** sklearn.metrics **import** accuracy\_score, balanced\_accuracy\_score

**from** sklearn.metrics **import** precision\_score, recall\_score, f1\_score, classification\_report

**from** sklearn.naive\_bayes **import** ComplementNB

**from** sklearn.metrics **import** confusion\_matrix

**from** sklearn.model\_selection **import** cross\_val\_score

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.metrics **import** mean\_absolute\_error, mean\_squared\_error, mean\_squared\_log\_error, median\_absolute\_error, r2\_score

**from** sklearn.metrics **import** roc\_curve, roc\_auc\_score

**from** sklearn.svm **import** SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**%matplotlib** inline

sns**.**set(style**=**"ticks")

In [ ]:

**def** accuracy\_score\_for\_classes(

y\_true: np**.**ndarray,

y\_pred: np**.**ndarray) **->** Dict[int, float]:

"""

Вычисление метрики accuracy для каждого класса

y\_true - истинные значения классов

y\_pred - предсказанные значения классов

Возвращает словарь: ключ - метка класса,

значение - Accuracy для данного класса

"""

*# Для удобства фильтрации сформируем Pandas DataFrame*

d **=** {'t': y\_true, 'p': y\_pred}

df **=** pd**.**DataFrame(data**=**d)

*# Метки классов*

classes **=** np**.**unique(y\_true)

*# Результирующий словарь*

res **=** dict()

*# Перебор меток классов*

**for** c **in** classes:

*# отфильтруем данные, которые соответствуют*

*# текущей метке класса в истинных значениях*

temp\_data\_flt **=** df[df['t']**==**c]

*# расчет accuracy для заданной метки класса*

temp\_acc **=** accuracy\_score(

temp\_data\_flt['t']**.**values,

temp\_data\_flt['p']**.**values)

*# сохранение результата в словарь*

res[c] **=** temp\_acc

**return** res

**def** print\_accuracy\_score\_for\_classes(

y\_true: np**.**ndarray,

y\_pred: np**.**ndarray):

"""

Вывод метрики accuracy для каждого класса

"""

accs **=** accuracy\_score\_for\_classes(y\_true, y\_pred)

**if** len(accs)**>**0:

print('Метка \t Accuracy')

**for** i **in** accs:

print('{} \t {}'**.**format(i, accs[i]))

In [ ]:

*# Загрузка данных*

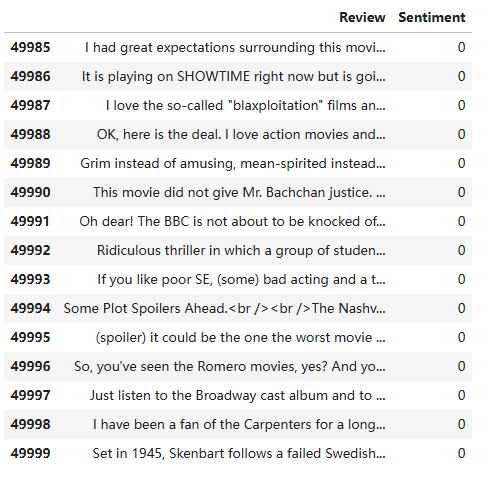
df **=** pd**.**read\_csv('D:\\Ботва\\Магистратура\\2сем\\ММО\\лаб6\\imdb\_sup.csv')

text\_df**=**df**.**head(500)**.**append(df**.**tail(500))

text\_df**.**drop('Rating', axis**=**1, inplace**=True**)

text\_df**.**tail(15)

Out[ ]:



In [ ]:

*#Кодирование целевого признаков*

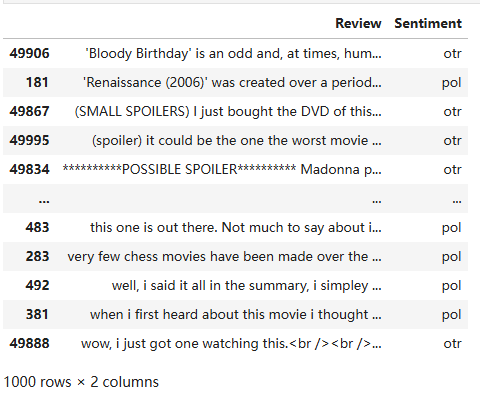
text\_df**.**loc[text\_df['Sentiment'] **==** 1, 'Sentiment'] **=** 'pol'

text\_df**.**loc[text\_df['Sentiment'] **==** 0, 'Sentiment'] **=** 'otr'

text\_df**.**sort\_values(by**=**['Review'], inplace**=True**)

text\_df

Out[ ]:



In [ ]:

text\_df**.**shape

Out[ ]:

(1000, 2)

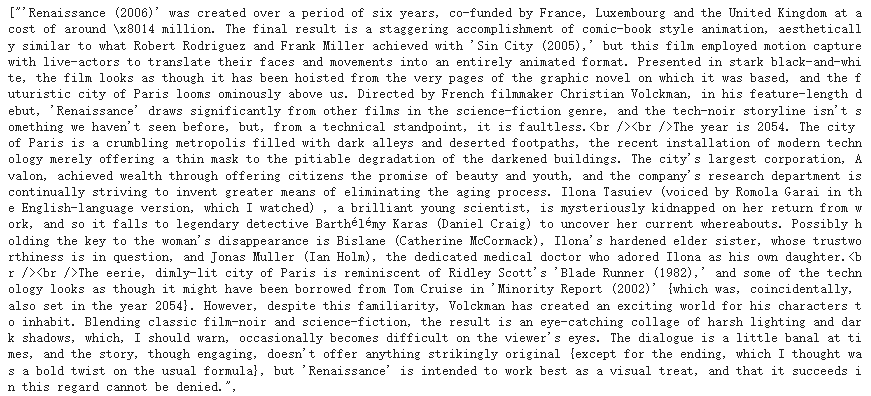
In [ ]:

*# Сформируем общий словарь для обучения моделей из обучающей и тестовой выборки*

vocab\_list **=** text\_df['Review']**.**tolist()

vocab\_list[1:10]

Out[ ]:



In [ ]:

vocabVect **=** CountVectorizer()

vocabVect**.**fit(vocab\_list)

corpusVocab **=** vocabVect**.**vocabulary\_

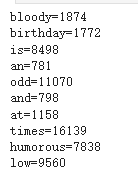
print('Количество сформированных признаков - {}'**.**format(len(corpusVocab)))

Количество сформированных признаков - 17896

In [ ]:

**for** i **in** list(corpusVocab)[0:10]:

print('{}={}'**.**format(i, corpusVocab[i]))



# Векторизация признаков на основе CountVectorizer

Подсчитывает количество слов словаря, входящих в данный текст

In [ ]:

test\_features **=** vocabVect**.**transform(vocab\_list)

In [ ]:

test\_features

Out[ ]:

<1000x17896 sparse matrix of type '<class 'numpy.int64'>'

with 137926 stored elements in Compressed Sparse Row format>

In [ ]:

test\_features**.**todense()

Out[ ]:

matrix([[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

...,

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0]], dtype=int64)

In [ ]:

*# Размер нулевой строки*

len(test\_features**.**todense()[0]**.**getA1())

Out[ ]:

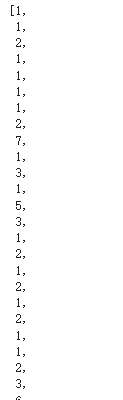
17896

In [ ]:

*# Непустые значения нулевой строки*

[i **for** i **in** test\_features**.**todense()[0]**.**getA1() **if** i**>**0]

Out[ ]:



In [ ]:

**def** VectorizeAndClassify(vectorizers\_list, classifiers\_list):

**for** v **in** vectorizers\_list:

**for** c **in** classifiers\_list:

pipeline1 **=** Pipeline([("vectorizer", v), ("classifier", c)])

score **=** cross\_val\_score(pipeline1, text\_df['Review'], text\_df['Sentiment'], scoring**=**'accuracy', cv**=**3)**.**mean()

print('Векторизация - {}'**.**format(v))

print('Модель для классификации - {}'**.**format(c))

print('Accuracy = {}'**.**format(score))

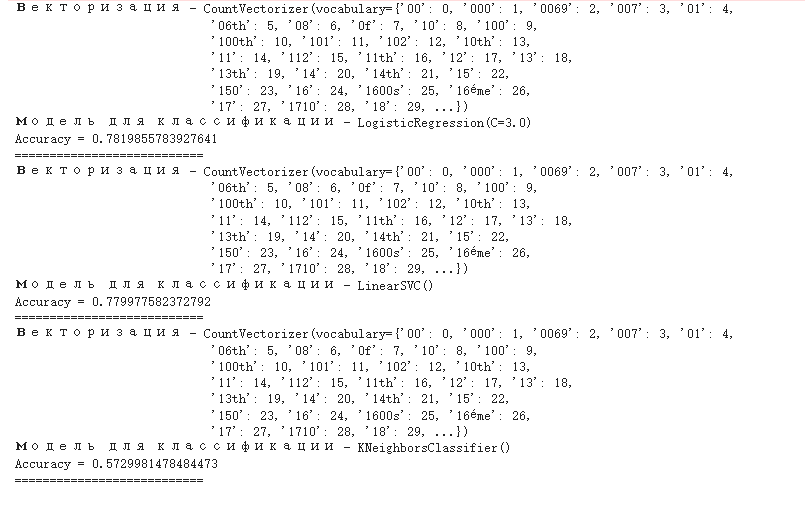
print('===========================')

In [ ]:

vectorizers\_list **=** [CountVectorizer(vocabulary **=** corpusVocab)]

classifiers\_list **=** [LogisticRegression(C**=**3.0), LinearSVC(), KNeighborsClassifier()]

VectorizeAndClassify(vectorizers\_list, classifiers\_list)



# Разделим на обучающую и тестовую выборки

In [ ]:

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(text\_df['Review'], text\_df['Sentiment'], test\_size**=**0.5, random\_state**=**1)

In [ ]:

**def** sentiment(v, c):

model **=** Pipeline(

[("vectorizer", v),

("classifier", c)])

model**.**fit(X\_train, y\_train)

y\_pred **=** model**.**predict(X\_test)

print\_accuracy\_score\_for\_classes(y\_test, y\_pred)

In [ ]:

sentiment(CountVectorizer(), LogisticRegression(C**=**3.0))



# Классификация текста на основе моделей word2vec

In [ ]:

**import** gensim

**from** gensim.models **import** word2vec

In [ ]:

**import** re

**from** typing **import** Dict, Tuple

**from** sklearn.metrics **import** accuracy\_score, balanced\_accuracy\_score

**from** sklearn.feature\_extraction.text **import** CountVectorizer, TfidfVectorizer

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.pipeline **import** Pipeline

**from** nltk **import** WordPunctTokenizer

**from** nltk.corpus **import** stopwords

**import** nltk

nltk**.**download('stopwords')

[nltk\_data] Downloading package stopwords to

[nltk\_data] C:\Users\sveta\AppData\Roaming\nltk\_data...

[nltk\_data] Package stopwords is already up-to-date!

Out[ ]:

True

In [ ]:

*# Подготовим корпус*

corpus **=** []

stop\_words **=** stopwords**.**words('english')

tok **=** WordPunctTokenizer()

**for** line **in** text\_df['Review']**.**values:

line1 **=** line**.**strip()**.**lower()

line1 **=** re**.**sub("[^a-zA-Z]"," ", line1)

text\_tok **=** tok**.**tokenize(line1)

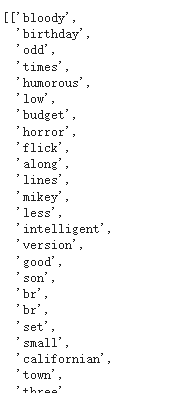
text\_tok1 **=** [w **for** w **in** text\_tok **if** **not** w **in** stop\_words]

corpus**.**append(text\_tok1)

In [ ]:

corpus[:10]

Out[ ]:



In [ ]:

**%time**

model\_imdb **=** word2vec**.**Word2Vec(corpus, workers**=**4, min\_count**=**10, window**=**10, sample**=**1e-3)

Wall time: 0 ns

In [ ]:

*# Проверим, что модель обучилась*

print(model\_imdb**.**wv**.**most\_similar(positive**=**['find'], topn**=**5))

[('someone', 0.9996957778930664), ('audience', 0.9996758103370667), ('far', 0.9996746182441711), ('everything', 0.9996718168258667), ('nothing', 0.9996710419654846)]

In [ ]:

**def** sentiment(v, c):

model **=** Pipeline(

[("vectorizer", v),

("classifier", c)])

model**.**fit(X\_train, y\_train)

y\_pred **=** model**.**predict(X\_test)

print\_accuracy\_score\_for\_classes(y\_test, y\_pred)

In [ ]:

**class** EmbeddingVectorizer(object):

'''

Для текста усредним вектора входящих в него слов

'''

**def** \_\_init\_\_(self, model):

self**.**model **=** model

self**.**size **=** model**.**vector\_size

**def** fit(self, X, y):

**return** self

**def** transform(self, X):

**return** np**.**array([np**.**mean(

[self**.**model[w] **for** w **in** words **if** w **in** self**.**model]

**or** [np**.**zeros(self**.**size)], axis**=**0)

**for** words **in** X])

In [ ]:

**def** accuracy\_score\_for\_classes(

y\_true: np**.**ndarray,

y\_pred: np**.**ndarray) **->** Dict[int, float]:

"""

Вычисление метрики accuracy для каждого класса

y\_true - истинные значения классов

y\_pred - предсказанные значения классов

Возвращает словарь: ключ - метка класса,

значение - Accuracy для данного класса

"""

*# Для удобства фильтрации сформируем Pandas DataFrame*

d **=** {'t': y\_true, 'p': y\_pred}

df **=** pd**.**DataFrame(data**=**d)

*# Метки классов*

classes **=** np**.**unique(y\_true)

*# Результирующий словарь*

res **=** dict()

*# Перебор меток классов*

**for** c **in** classes:

*# отфильтруем данные, которые соответствуют*

*# текущей метке класса в истинных значениях*

temp\_data\_flt **=** df[df['t']**==**c]

*# расчет accuracy для заданной метки класса*

temp\_acc **=** accuracy\_score(

temp\_data\_flt['t']**.**values,

temp\_data\_flt['p']**.**values)

*# сохранение результата в словарь*

res[c] **=** temp\_acc

**return** res

**def** print\_accuracy\_score\_for\_classes(

y\_true: np**.**ndarray,

y\_pred: np**.**ndarray):

"""

Вывод метрики accuracy для каждого класса

"""

accs **=** accuracy\_score\_for\_classes(y\_true, y\_pred)

**if** len(accs)**>**0:

print('Метка \t Accuracy')

**for** i **in** accs:

print('{} \t {}'**.**format(i, accs[i]))

In [ ]:

*# Обучающая и тестовая выборки*

boundary **=** 900

X\_train **=** corpus[:boundary]

X\_test **=** corpus[boundary:]

y\_train **=** text\_df['Sentiment'][:boundary]

y\_test **=** text\_df['Sentiment'][boundary:]

In [ ]:

sentiment(EmbeddingVectorizer(model\_imdb**.**wv), LogisticRegression(C**=**3.0))

