

Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»



Лабораторная работа №9
по дисциплине
«Методы машинного обучения»
на тему
«Классификация текста»

Выполнил:
студент группы ИУ5И-22М
Лу Жуньда

Москва — 2024 г.

1. Цель лабораторной работы

Изучение методов классификации текстов.

2. Задание

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

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

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

3. Текст программы

1. Установка и импорт необходимых библиотек

```
# Установка необходимых библиотек
!pip install numpy pandas scikit-learn gensim kaggle

# Импорт необходимых библиотек
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import gensim.downloader as api
from gensim.models import Word2Vec
from sklearn.ensemble import RandomForestClassifier
```

```
# Аутентификация Kaggle
from google.colab import files
files.upload()

# Создание kaggle директории и копирование kaggle.json
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Загрузка набора данных
!kaggle datasets download -d clmentbisaillon/fake-and-real-news-dataset

# Распаковка набора данных
!unzip fake-and-real-news-dataset.zip
```

选择文件 kaggle.json

- **kaggle.json**(application/json) - 63 bytes, last modified: 2024/6/6 - 100% done

Saving kaggle.json to kaggle.json

Dataset URL: <https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset>

License(s): CC-BY-NC-SA-4.0

Downloading fake-and-real-news-dataset.zip to /content

81% 33.0M/41.0M [00:00<00:00, 102MB/s]

100% 41.0M/41.0M [00:00<00:00, 106MB/s]

Archive: fake-and-real-news-dataset.zip

inflating: Fake.csv

inflating: True.csv

```

# Чтение данных
df_fake = pd.read_csv('Fake.csv')
df_real = pd.read_csv('True.csv')

# Добавление меток классов
df_fake['label'] = 0
df_real['label'] = 1

# Объединение данных
df = pd.concat([df_fake, df_real])

# Перемешивание данных
df = df.sample(frac=1).reset_index(drop=True)

# Разделение данных на признаки и метки
texts = df['text'].values
labels = df['label'].values

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_state=42)

```

✓ Способ 1: Классификация на основе CountVectorizer или TfidfVectorizer

Использование CountVectorizer

```

[6] # Преобразование текстов в векторное представление
count_vectorizer = CountVectorizer()
X_train_counts = count_vectorizer.fit_transform(X_train)
X_test_counts = count_vectorizer.transform(X_test)

# Обучение модели
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_counts, y_train)

# Предсказание
y_pred_counts = nb_classifier.predict(X_test_counts)

# Оценка качества модели
print("Accuracy (CountVectorizer):", accuracy_score(y_test, y_pred_counts))
print("Classification Report (CountVectorizer):\n", classification_report(y_test, y_pred_counts))

```

```

⇒ Accuracy (CountVectorizer): 0.9540089086859688
Classification Report (CountVectorizer):

```

	precision	recall	f1-score	support
0	0.96	0.95	0.96	4693
1	0.95	0.95	0.95	4287
accuracy			0.95	8980
macro avg	0.95	0.95	0.95	8980
weighted avg	0.95	0.95	0.95	8980

✓ Использование TfidfVectorizer

```
[7] # Преобразование текстов в векторное представление
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Обучение модели
nb_classifier.fit(X_train_tfidf, y_train)

# Предсказание
y_pred_tfidf = nb_classifier.predict(X_test_tfidf)

# Оценка качества модели
print("Accuracy (TfidfVectorizer):", accuracy_score(y_test, y_pred_tfidf))
print("Classification Report (TfidfVectorizer):\n", classification_report(y_test, y_pred_tfidf))
```

```
⇒ Accuracy (TfidfVectorizer): 0.9364142538975501
Classification Report (TfidfVectorizer):
      precision    recall  f1-score   support

     0       0.93      0.95      0.94       4693
     1       0.94      0.92      0.93       4287

 accuracy
macro avg       0.94      0.94      0.94       8980
weighted avg       0.94      0.94      0.94       8980
```

✓ Способ 2: Классификация на основе моделей word2vec

Обучение модели word2vec

```
[8] # Загрузка предобученной модели word2vec
word2vec_model = api.load("word2vec-google-news-300")

# Преобразование текстов в векторное представление
def text_to_word2vec(text, model):
    words = text.split()
    word_vecs = [model[word] for word in words if word in model]
    return np.mean(word_vecs, axis=0) if len(word_vecs) > 0 else np.zeros(model.vector_size)

X_train_word2vec = np.array([text_to_word2vec(text, word2vec_model) for text in X_train])
X_test_word2vec = np.array([text_to_word2vec(text, word2vec_model) for text in X_test])

# Обучение модели
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train_word2vec, y_train)

# Предсказание
y_pred_word2vec = rf_classifier.predict(X_test_word2vec)

# Оценка качества модели
print("Accuracy (word2vec):", accuracy_score(y_test, y_pred_word2vec))
print("Classification Report (word2vec):\n", classification_report(y_test, y_pred_word2vec))
```

 [=====] 100.0% 1662.8/1662.8MB downloaded

Accuracy (word2vec): 0.9687082405345212


Classification Report (word2vec):

	precision	recall	f1-score	support
0	0.97	0.97	0.97	4693
1	0.97	0.97	0.97	4287
accuracy			0.97	8980
macro avg	0.97	0.97	0.97	8980
weighted avg	0.97	0.97	0.97	8980

✓ Сравнение качества моделей

Вывод результатов классификации и сравнение метрик позволит оценить, какой метод оказался более эффективным для данного набора данных.

```
[11] print("Summary of Results:")
print("CountVectorizer Accuracy:", accuracy_score(y_test, y_pred_counts))
print("TfidfVectorizer Accuracy:", accuracy_score(y_test, y_pred_tfidf))
print("Word2Vec Accuracy:", accuracy_score(y_test, y_pred_word2vec))
```

 Summary of Results:
CountVectorizer Accuracy: 0.954008906659688
TfidfVectorizer Accuracy: 0.9364142538975501
Word2Vec Accuracy: 0.9687082405345212

Заключение

В данной лабораторной работе мы рассмотрели два различных подхода к классификации текстов и сравнили их результаты. Вот сводка результатов:

CountVectorizer:

Точность: 95.40%

Методы на основе CountVectorizer показали хорошую точность, демонстрируя, что частотные характеристики слов могут эффективно использоваться для классификации текстов.

TfidfVectorizer:

Точность: 93.64%

Методы на основе TfidfVectorizer также показали высокую точность, хотя немного ниже по сравнению с CountVectorizer. Использование TF-IDF позволяет учесть важность слов в контексте всего корпуса текстов, что также является мощным инструментом для классификации.

Word2Vec:

Точность: 96.87%

Методы на основе Word2Vec показали наивысшую точность среди рассмотренных методов. Это может быть связано с тем, что Word2Vec учитывает семантические связи между словами, что позволяет более точно представлять тексты и улучшать качество классификации.