МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №__9__
по дисциплине «Методы машинного обучения в автоматизированных системах»

Тема: «Классификация текста»

ИСПОЛНИТЕЛЬ: ___Калюта Н.И.___
ФИО
группа ИУ5-22М _____
подпись
"30"___05___2024 г.

ПРЕПОДАВАТЕЛЬ: ___ _____
ФИО
_____
подпись
"__"_____2024 г.

Москва - 2024

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.

Импортирование необходимых библиотек

```
[32] import numpy as np
     import pandas as pd
     import gensim
     from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
     from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.svm import SVC, LinearSVC
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score
     import time
     import gensim.downloader as api
     from tqdm import tqdm
     import sys
```

Подключение гугл диска

```
[33] from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[34] # Загрузка данных
     train_data = pd.read_csv('/content/drive/MyDrive/RK/Electric_train.csv', encoding='latin1')
     test_data = pd.read_csv('/content/drive/MyDrive/RK/Electric_test.csv', encoding='latin1')
```

```
[35] train_data.shape
```

```
(149503, 17)
```

```
train_data.head()
```

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility
0	WBY8P6C58K	King	Seattle	WA	98115.0	2019.0	BMW	I3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible
1	5YJSA1DN4D	Kitsap	Bremerton	WA	98312.0	2013.0	TESLA	MODEL S	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible
2	5YJSA1E26J	King	Kent	WA	98042.0	2018.0	TESLA	MODEL S	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible
3	WBY2Z2C54E	King	Bellevue	WA	98004.0	2014.0	BMW	I8	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range
4	5YJXCDE23J	King	Bellevue	WA	98004.0	2018.0	TESLA	MODEL X	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible

```
[37] test_data.shape
```

```
(37376, 17)
```

```
[38] test_data.head()
```

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility
0	1GYKPSRL2R	Spokane	Newman Lake	WA	99025.0	2024.0	CADILLAC	LYRIQ	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...
1	KM8KRDAF1P	Snohomish	Mountlake Terrace	WA	98043.0	2023.0	HYUNDAI	IONIQ 5	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...
2	7SAYGDEF3P	King	Seattle	WA	98101.0	2023.0	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...
3	7SAYGDEE6P	King	Newcastle	WA	98056.0	2023.0	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...
4	JTMEB3FV3M	San Juan	Friday Harbor	WA	98250.0	2021.0	TOYOTA	RAV4 PRIME	Plug-in Hybrid Electric Vehicle (PHEV)	Clean Alternative Fuel Vehicle Eligible

```
[8]  train_data.dropna(inplace=True)
     test_data.dropna(inplace=True)
```

```
[9]  X_train = train_data['City']
     y_train = train_data['Electric Vehicle Type']

     X_test = test_data['City']
     y_test = test_data['Electric Vehicle Type']
```

```
def check_missing(data, name):
    missing = data.isnull().sum()
    print(f'У {name} {missing} пропущенных строк')
```

```
[11] check_missing(train_data, 'train_data')
     check_missing(test_data, 'test_data')
     check_missing(X_train, 'X_train')
     check_missing(X_test, 'X_test')
     check_missing(y_train, 'y_train')
     check_missing(y_test, 'y_test')
```

```
У train_data VIN (1-10)                                0
County                                                 0
City                                                   0
State                                                  0
Postal Code                                            0
Model Year                                             0
Make                                                   0
Model                                                  0
Electric Vehicle Type                                  0
Clean Alternative Fuel Vehicle (CAFV) Eligibility      0
Electric Range                                         0
Base MSRP                                              0
Legislative District                                   0
DOL Vehicle ID                                         0
Vehicle Location                                       0
Electric Utility                                       0
2020 Census Tract                                      0
dtype: int64 пропущенных строк
У test_data VIN (1-10)                                 0
County                                                 0
City                                                   0
State                                                  0
Postal Code                                            0
Model Year                                             0
Make                                                   0
Model                                                  0
Electric Vehicle Type                                  0
Clean Alternative Fuel Vehicle (CAFV) Eligibility      0
Electric Range                                         0
Base MSRP                                              0
Legislative District                                   0
DOL Vehicle ID                                         0
Vehicle Location                                       0
Electric Utility                                       0
2020 Census Tract                                      0
dtype: int64 пропущенных строк
У X_train 0 пропущенных строк
У X_test 0 пропущенных строк
У y_train 0 пропущенных строк
У y_test 0 пропущенных строк
```

```python
# Векторизация с помощью CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
X_test_counts = count_vect.transform(X_test)

# Векторизация с помощью TfidfVectorizer
tfidf_vect = TfidfVectorizer()
X_train_tfidf = tfidf_vect.fit_transform(X_train)
X_test_tfidf = tfidf_vect.transform(X_test)
```

```python
# Загрузка предобученных моделей
word2vec_model = api.load('word2vec-google-news-300')
glove_model = api.load('glove-twitter-200')
fasttext_model = api.load('fasttext-wiki-news-subwords-300')
```

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
[==================================================] 100.0% 758.5/758.5MB downloaded
[==================================================] 100.0% 958.5/958.4MB downloaded
```

```python
# Функция для усреднения векторов слов в тексте
def vectorize_text(text, model):
    words = text.split()
    vectors = [model[word] for word in words if word in model]
    if len(vectors) == 0:
        return np.zeros(model.vector_size)
    return np.mean(vectors, axis=0)

# Векторизация данных
def vectorize_dataset(dataset, model, desc="Vectorizing"):
    return np.array([vectorize_text(text, model) for text in tqdm(dataset, desc=desc)])
```

```python
X_train_w2v = vectorize_dataset(X_train, word2vec_model, desc="Vectorizing word2vec")
X_test_w2v = vectorize_dataset(X_test, word2vec_model, desc="Vectorizing word2vec")

X_train_glove = vectorize_dataset(X_train, glove_model, desc="Vectorizing Glove")
X_test_glove = vectorize_dataset(X_test, glove_model, desc="Vectorizing Glove")

X_train_fasttext = vectorize_dataset(X_train, fasttext_model, desc="Vectorizing fastText")
X_test_fasttext = vectorize_dataset(X_test, fasttext_model, desc="Vectorizing fastText")
```

```
Vectorizing word2vec: 100%|██████████| 149060/149060 [00:06<00:00, 23806.33it/s]
Vectorizing word2vec: 100%|██████████| 36992/36992 [00:02<00:00, 17743.55it/s]
Vectorizing Glove: 100%|██████████| 149060/149060 [00:04<00:00, 34877.50it/s]
Vectorizing Glove: 100%|██████████| 36992/36992 [00:00<00:00, 47731.49it/s]
Vectorizing fastText: 100%|██████████| 149060/149060 [00:06<00:00, 24584.65it/s]
Vectorizing fastText: 100%|██████████| 36992/36992 [00:02<00:00, 18377.05it/s]
```

```python
# Функции для оценки точности для каждой метки
def accuracy_score_for_classes(y_true, y_pred):
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    classes = np.unique(y_true)
    res = dict()
    for c in classes:
        temp_data_flt = df[df['t'] == c]
        temp_acc = accuracy_score(temp_data_flt['t'].values, temp_data_flt['p'].values)
        res[c] = temp_acc
    return res
```

```python
def print_accuracy_score_for_classes(y_true, y_pred):
    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]))
```

```python
# Оценка моделей
def evaluate_model(vectorizer_name, vectorizer_train, vectorizer_test, model, model_name):
    start_time = time.time()
    obj_model = model
    obj_model.fit(vectorizer_train, y_train)
    predictions = obj_model.predict(vectorizer_test)

    accuracy = accuracy_score(y_test, predictions)
    duration = (time.time() - start_time) / 60

    print(f'Точность: {accuracy:.4f}, время обучения классификатора: {duration:.2f} мин. ({vectorizer_name} + {model_name})')
    print_accuracy_score_for_classes(y_test, predictions)
```

```
[50] classifiers = {
         "RandomForestClassifier": RandomForestClassifier(),
         "LinearSVC": LinearSVC(max_iter=10000),
         "LogisticRegression": LogisticRegression(max_iter=10000)
     }

     vectorizers = {
         "CountVectorizer": (X_train_counts, X_test_counts),
         "TfidfVectorizer": (X_train_tfidf, X_test_tfidf),
         "word2vec": (X_train_w2v, X_test_w2v),
         "glove": (X_train_glove, X_test_glove),
         "fastText": (X_train_fasttext, X_test_fasttext)
     }
```

```
[20] for vec_name, (train_vec, test_vec) in vectorizers.items():
         for clf_name, clf in classifiers.items():
             evaluate_model(vec_name, train_vec, test_vec, clf, clf_name)
```

```
Точность: 0.7792, время обучения классификатора: 0.24 мин. (CountVectorizer + RandomForestClassifier)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9950500190383883
Plug-in Hybrid Electric Vehicle (PHEV)   0.009749475502900161
Точность: 0.7793, время обучения классификатора: 0.03 мин. (CountVectorizer + LinearSVC)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9955346325591056
Plug-in Hybrid Electric Vehicle (PHEV)   0.008515364679748241
Точность: 0.7794, время обучения классификатора: 0.03 мин. (CountVectorizer + LogisticRegression)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9961923223372218
Plug-in Hybrid Electric Vehicle (PHEV)   0.006417376280389979
Точность: 0.7790, время обучения классификатора: 0.26 мин. (TfidfVectorizer + RandomForestClassifier)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9946692512721105
Plug-in Hybrid Electric Vehicle (PHEV)   0.009996297667530544
Точность: 0.7793, время обучения классификатора: 0.02 мин. (TfidfVectorizer + LinearSVC)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9955346325591056
Plug-in Hybrid Electric Vehicle (PHEV)   0.008515364679748241
Точность: 0.7794, время обучения классификатора: 0.02 мин. (TfidfVectorizer + LogisticRegression)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9961923223372218
Plug-in Hybrid Electric Vehicle (PHEV)   0.006417376280389979
Точность: 0.7789, время обучения классификатора: 1.30 мин. (word2vec + RandomForestClassifier)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9947384817750701
Plug-in Hybrid Electric Vehicle (PHEV)   0.009255831173639394
Точность: 0.7794, время обучения классификатора: 1.54 мин. (word2vec + LinearSVC)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9951538647928277
Plug-in Hybrid Electric Vehicle (PHEV)   0.009996297667530544
Точность: 0.7792, время обучения классификатора: 0.60 мин. (word2vec + LogisticRegression)
Метка    Accuracy
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9957423240679844
Plug-in Hybrid Electric Vehicle (PHEV)   0.0074046649389115145
Точность: 0.7810, время обучения классификатора: 0.37 мин. (glove + RandomForestClassifier)
Метка    Accuracy
Battery Electric Vehicle (BEV)   1.0
Plug-in Hybrid Electric Vehicle (PHEV)   0.0
Точность: 0.7810, время обучения классификатора: 0.01 мин. (glove + LinearSVC)
Метка    Accuracy
Battery Electric Vehicle (BEV)   1.0
Plug-in Hybrid Electric Vehicle (PHEV)   0.0
Точность: 0.7810, время обучения классификатора: 0.01 мин. (glove + LogisticRegression)
Метка    Accuracy
Battery Electric Vehicle (BEV)   1.0
Plug-in Hybrid Electric Vehicle (PHEV)   0.0
Точность: 0.7790, время обучения классификатора: 1.12 мин. (fastText + RandomForestClassifier)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9949115580324691
Plug-in Hybrid Electric Vehicle (PHEV)   0.009255831173639394
Точность: 0.7793, время обучения классификатора: 0.34 мин. (fastText + LinearSVC)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.995811554570944
Plug-in Hybrid Electric Vehicle (PHEV)   0.007528076021226706
Точность: 0.7794, время обучения классификатора: 0.30 мин. (fastText + LogisticRegression)
Метка    Accuracy
Battery Electric Vehicle (BEV)   0.9962615528401814
Plug-in Hybrid Electric Vehicle (PHEV)   0.006293965198074787
```

Вывод:

В ходе данной работы были протестированы различные комбинации методов векторизации текста (CountVectorizer, TfidfVectorizer, word2vec, GloVe, fastText) и алгоритмов классификации (RandomForestClassifier, LinearSVC, LogisticRegression) для задачи

классификации типов электрических транспортных средств. Наилучшая точность классификации составила 78.10% и была достигнута при использовании метода векторизации GloVe в сочетании с любым из трех рассмотренных алгоритмов классификации. Модели показали высокую точность классификации для класса "Battery Electric Vehicle (BEV)", но более низкую точность для класса "Plug-in Hybrid Electric Vehicle (PHEV)". Полученные результаты демонстрируют, что комбинация методов векторизации текста и алгоритмов классификации может оказывать существенное влияние на качество решения задачи классификации.