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 «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №4 по дисциплине "Методы машинного обучения"

Тема Наивный байесовский классификатор для фильтрации спама

Студент Варламова Е. А.

Группа ИУ7-23М

Оценка (баллы) _____

Преподаватели Солодовников Владимир Игоревич

Москва — 2024 г.

СОДЕРЖАНИЕ

1 | Теоретическая часть

Фильтрация спама является необходимой для обеспечения безопасности и эффективности работы в сети. Спам может содержать вредоносные ссылки, вирусы или фишинговые атаки, которые могут нанести ущерб вашим данным или устройствам. Кроме того, спам может засорять почтовый ящик или другие коммуникационные каналы, делая их менее эффективными для передачи важных сообщений. Фильтрация спама помогает защитить вас от нежелательного контента и обеспечивает более продуктивное использование интернета и электронной почты.

Целью данной лабораторной работы является применение наивного байесовского классификатора для фильтрации спама. Для этого необходимо решить следующие задачи:

— формализовать задачу;

— описать алгоритм наивного байесовского классификатора;

— привести схему ПО, реализующего фильтрацию спама с помощью наивного байесовского классификатора;

— привести особенности реализации ПО, решающего поставленную задачу;

— провести исследование зависимости результатов точности классификации в зависимости от алгоритма векторизации, применяемого до классификации.

1.1 Постановка задачи

Решить задачу фильтрации спама с использованием наивного байесовского классификатора, основанного на принципе максимума апостериорной вероятности. В качестве исходных данных можно использовать данные с сайта Kaggle (SMS Spam Collection Dataset) (можно какой-то другой датасет). В выполнения работы:

1. Осуществить предобработку данных. Сформировать обучающий и тестовый наборы данных.

2. Построить наивный байесовский классификатор.

3. Оценить качество работы классификатора.

1.2 Алгоритм наивного байесовского классификатора

Классификация с использованием наивного байесовского классификатора основана на применении теоремы Байеса. Пусть дан набор объектов $X = (x_1, x_2, ..., x_n)$ и метки классов $Y = (y_1, y_2, ..., y_m)$. Наивный байесовский классификатор предполагает, что все признаки объектов независимы друг от друга при условии класса. Таким образом, вероятность принадлежности объекта x к классу y может быть вычислена по формуле:

$$P(y|x) = P(x|y)P(y)/P(x) \tag{1.1}$$

где

1. $P(y|x)$ - вероятность принадлежности объекта x к классу y;

2. $P(x|y)$ - вероятность объекта x при условии класса y;

3. $P(y)$ - априорная вероятность класса y;

4. $P(x)$- вероятность объекта x.

Для наивного байесовского классификатора используются различные модели для оценки вероятностей $P(x|y)$ и $P(y)$. Например, для категориальных данных можно использовать модель мультиномиального распределения, а для непрерывных данных - модель нормального распределения.

На практике для классификации объекта выбирается класс с наибольшей вероятностью $P(y|x)$:

$$y' = max_y \in YP(y|x) \tag{1.2}$$

где y' - предсказанная метка класса для объекта x.

Таким образом, наивный байесовский классификатор является простым и эффективным методом классификации, основанным на принципах вероятностного вывода.

1.3 Алгоритм векторизации CountVectorizer

CountVectorizer – это метод векторизации текстовых данных, который преобразует текстовые данные в матрицу частот слов. Алгоритм работы CountVectorizer можно описать следующим образом:

1. Токенизация: Исходный текст разбивается на отдельные слова или токены. Токенизация может проводиться с использованием различных методов, например, разделение по пробелам или использование регулярных выражений.

2. Построение словаря: CountVectorizer строит словарь всех уникальных слов (токенов) в исходном тексте. Каждому уникальному слову присваивается уникальный индекс.

3. Подсчет частот слов: Для каждого текста подсчитывается количество вхождений каждого слова из словаря. Эти частоты слов образуют строки матрицы частот слов.

4. Преобразование векторов: Каждый текст представляется в виде вектора, где каждый элемент соответствует количеству вхождений соответствующего слова из словаря.

5. Преобразование редких матриц: Поскольку большинство слов в тексте редко встречаются, матрица частот слов будет разреженной. CountVectorizer может сохранять разреженные матрицы для экономии памяти.

1.4 Алгоритм векторизации TfidfVectorizer

TfidfVectorizer - это метод векторизации текстовых данных, который преобразует текстовые данные в матрицу TF-IDF (Term Frequency-Inverse Document Frequency). Алгоритм работы TfidfVectorizer можно описать следующим образом:

1. Токенизация: Исходный текст разбивается на отдельные слова или токены. Токенизация может проводиться с использованием различных методов, например, разделение по пробелам или использование регулярных выражений.

2. Подсчет TF (Term Frequency): Для каждого текста подсчитывается количество вхождений каждого слова из словаря. Это называется частотой слова

в документе (TF). TF показывает, насколько часто слово встречается в данном документе.

3. Подсчет IDF (Inverse Document Frequency): IDF вычисляется как логарифм отношения общего числа документов к числу документов, содержащих данное слово. IDF показывает, насколько информативно слово является для всех документов.

4. Вычисление TF-IDF: TF-IDF для каждого слова в документе вычисляется как произведение TF и IDF для этого слова. Это дает вес словам, учитывая как их частоту в текущем документе, так и их уникальность во всей коллекции документов.

5. Преобразование векторов: Каждый текст представляется в виде вектора, где каждый элемент соответствует значению TF-IDF для соответствующего слова из словаря.

6. Нормализация векторов: Векторы могут быть нормализованы для улучшения работы моделей машинного обучения.

TfidfVectorizer помогает учитывать не только частоту слова в документе, но и его важность для всей коллекции текстов. Этот метод широко используется в задачах анализа текста, кластеризации и рекомендательных системах.

1.5 Схема работы ПО

На рисунке 1.1 представлена схема работы ПО.

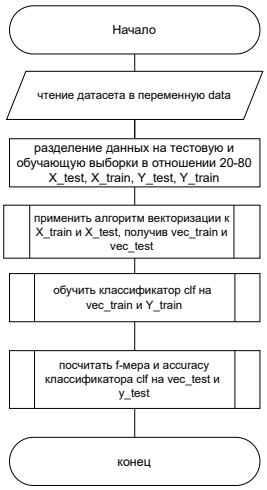

Рис. 1.1: схема работы ПО

2 | Практическая часть

2.1 Выбор средств разработки

В качестве языка программирования был использован язык Python, поскольку этот язык кроссплатформенный и для него разработано огромное количество библиотек и модулей, решающих разнообразные задачи.

В частности, имеются библиотеки, включающие в себя алгоритм наивного байесовского классификатора, а также алгоритмы векторизации текстов в библиотеке [1].

2.2 Исследование ПО

В листинге 2.1 представлен код, решающий задачу классификации спама.

Листинг 2.1: Код классификации спама

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix,
    precision_recall_fscore_support, accuracy_score
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from pymorphy2 import MorphAnalyzer
import warnings

warnings.filterwarnings("ignore")
morph = MorphAnalyzer()
stop_words = set(stopwords.words('english'))
lemma = True

def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    if lemma:
        tokens = [morph.parse(token)[0].normal_form for token in tokens]
    return ' '.join(tokens)
```

```
22
23 def CountVectorization(X_train, X_test):
24     vectorizer = CountVectorizer()
25     X_train_counts = vectorizer.fit_transform(X_train)
26     X_test_counts = vectorizer.transform(X_test)
27     return X_train_counts, X_test_counts
28 def tfidfVectorization(X_train, X_test):
29     tfidf_vectorizer = TfidfVectorizer()
30     X_train_counts = tfidf_vectorizer.fit_transform(X_train.apply(
           preprocess_text))
31     X_test_counts = tfidf_vectorizer.transform(X_test.apply(preprocess_text)
           )
32     return X_train_counts, X_test_counts
33
34 data = pd.read_csv('spam.csv', encoding='latin-1')
35 data = data[['v1', 'v2']]
36 data.columns = ['label', 'text']
37
38 data['label'] = data['label'].map({'ham': 0, 'spam': 1})
39 X = data['text']
40 y = data['label']
41
42 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
       random_state=42)
43 vecs = ["CountVectorization", "tfidfVectorization"]
44 i = 0
45 for vectorization in [CountVectorization, tfidfVectorization]:
46     print(vecs[i])
47     X_train_counts, X_test_counts = vectorization(X_train, X_test)
48
49     clf = MultinomialNB()
50     clf.fit(X_train_counts, y_train)
51
52     y_pred = clf.predict(X_test_counts)
53     accuracy = accuracy_score(y_test, y_pred)
54     precision, recall, f1, _ = precision_recall_fscore_support(y_test,
           y_pred, average='weighted')
55     print("Accuracy: ", accuracy)
56     print("f-        :", f1)
57     cm = confusion_matrix(y_test, y_pred)
58     print(f"                              :")
59     print(cm)
60     print()
61     i += 1
```

С помощью разработанного ПО было проведено исследование зависимости точности классификации от алгоритма предобработки текстов (алгоритма векторизации текстов).

Таблица 2.1: Результаты работы ПО

Алгоритм векторизации	Точность (accuracy)	Точность (f-мера)	Матрица ошибок
CountVectorization	0.98386	0.98352	[[963 2] [16 134]]
tfidfVectorization	0.96233	0.95967	[[965 0] [42 108]]

Видим, что более простой алгоритм векторизации в паре с наивным байесовским классификатором показал лучший результат классификации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Virtanen P.*, *Gommers R.*, *Oliphant T. E.* SciPy: Fundamental Algorithms for Scientific Computing in Python. — 2020. — DOI: 10.1038/s41592-019-0686-2.