**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В. И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Информатика»**

**Тема: «Введение в анализ данных»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Колесниченко М. А. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Изучить основные принципы анализа данных и освоить ключевые инструменты для их обработки и анализа. Овладеть навыками работы с данными, включая сбор, очистку и интерпретацию результатов.

## Задание

Вариант 1.

Вы работаете в магазине элитных вин и собираетесь провести анализ существующего ассортимента, проверив возможности инструмента классификации данных для выделения различных классов вин.

Для этого необходимо использовать библиотеку sklearn и встроенный в него набор данных о вине.

1) Загрузка данных:

Реализуйте функцию load\_data(), принимающей на вход аргумент train\_size (размер обучающей выборки, по умолчанию равен 0.8), которая загружает набор данных о вине из библиотеки sklearn в переменную wine. Разбейте данные для обучения и тестирования в соответствии со значением train\_size, следующим образом: из данного набора запишите train\_size данных из data, взяв при этом только 2 столбца в переменную X\_train и train\_size данных поля target в y\_train. В переменную X\_test положите оставшуюся часть данных из data, взяв при этом только 2 столбца, а в y\_test — оставшиеся данные поля target, в этом вам поможет функция train\_test\_split модуля sklearn.model\_selection ( в качестве состояния рандомизатора функции train\_test\_split необходимо указать 42.).

В качестве результата верните X\_train, X\_test, y\_train, y\_test.

Пояснение: X\_train, X\_test - двумерный массив, y\_train, y\_test. — одномерный массив.

2) Обучение модели. Классификация методом k-ближайших соседей:

Реализуйте функцию train\_model(), принимающую обучающую выборку (два аргумента - X\_train и y\_train) и аргументы n\_neighbors и weights (значения по умолчанию 15 и 'uniform' соответственно), которая создает экземпляр классификатора KNeighborsClassifier и загружает в него данные X\_train, y\_train c параметрами n\_neighbors и weights.

В качестве результата верните экземпляр классификатора.

3) Применение модели. Классификация данных

Реализуйте функцию predict(), принимающую обученную модель классификатора и тренировочный набор данных (X\_test), которая выполняет классификацию данных из X\_test.

В качестве результата верните предсказанные данные.

4) Оценка качества полученных результатов классификации.

Реализуйте функцию estimate(), принимающую результаты классификации и истинные метки тестовых данных (y\_test), которая считает отношение предсказанных результатов, совпавших с «правильными» в y\_test к общему количеству результатов. (или другими словами, ответить на вопрос «На сколько качественно отработала модель в процентах»).

В качестве результата верните полученное отношение, округленное до 0,001. В отчёте приведите объяснение полученных результатов.

Пояснение: так как это вероятность, то ответ должен находиться в диапазоне [0, 1].

5) Забытая предобработка:

После окончания рабочего дня перед сном вы вспоминаете лекции по предобработке данных и понимаете, что вы её не сделали...

Реализуйте функцию scale(), принимающую аргумент, содержащий данные, и аргумент mode - тип скейлера (допустимые значения: 'standard', 'minmax', 'maxabs', для других значений необходимо вернуть None в качестве результата выполнения функции, значение по умолчанию - 'standard'), которая обрабатывает данные соответствующим скейлером.

В качестве результата верните полученные после обработки данные.

## Выполнение работы

Функция `load\_data()` загружает набор данных о винах из библиотеки sklearn.datasets и разделяет его на тренировочные и тестовые выборки. Сначала загружается набор данных о винах из библиотеки sklearn.datasets, затем с помощью функции `train\_test\_split` данные разделяются на тренировочные и тестовые выборки в соответствии с заданным размером `train\_size` (по умолчанию равным 0.8). Результатом работы функции являются тренировочные и тестовые данные.

Функция `train\_model()` обучает модель классификации методом k-ближайших соседей (k-Nearest Neighbors, KNN) на предоставленных тренировочных данных. Сначала создается экземпляр классификатора KNN с заданными параметрами `n\_neighbors` и `weights`. Затем, с помощью функции `fit()`, модель обучается на тренировочных данных `X\_train` и соответствующих метках `y\_train`, после чего возвращает обученную модель.

Функция `predict()` выполняет прогнозирование классов для тестовых данных с использованием обученной модели классификатора. С помощью метода `predict()` обученной модели выполняется прогнозирование классов для переданных данных `X\_test`.

Функция `estimate()` оценивает точность модели классификации, сравнивая предсказанные метки классов с истинными метками тестового набора данных. Методом `accuracy\_score()` вычисляется точность модели, путем сравнения истинных меток класса `y\_test` с предсказанными метками `res`. Результат округляется до трех знаков после запятой.

Функция `scale()` выполняет масштабирование данных, используя один из нескольких способов нормализации, в зависимости от переданного режима. С помощью конструкции if-elif-else выбирается соответствующий метод масштабирования в зависимости от значения `mode`, затем с помощью метода `scaler.fit\_transform(data)` выбранный метод масштабирования применяется к переданным данным.

Разработанный программный код см. в приложении А.

## Тестирование

Таблица 1 - Исследование работы классификатора, обученного на данных разного размера

|  |  |  |
| --- | --- | --- |
| № | train\_size | accuracy |
| 1. | 0.1 | 0.778 |
| 2. | 0.3 | 0.839 |
| 3. | 0.5 | 0.889 |
| 4. | 0.7 | 0.944 |
| 5. | 0.9 | 0.972 |

Таблица 2 - Исследование работы классификатора, обученного с различными значениями n\_neighbors

|  |  |  |
| --- | --- | --- |
| № | n\_neighbors | accuracy |
| 1. | 3 | 0.944 |
| 2. | 5 | 0.972 |
| 3. | 9 | 0.972 |
| 4. | 15 | 0.972 |
| 5. | 25 | 0.944 |

Таблица 3 - Исследование работы классификатора с предобработанными данными

|  |  |  |
| --- | --- | --- |
| № | sacler | accuracy |
| 1. | StandardScaler | 0.972 |
| 2. | MinMaxScaler | 0.972 |
| 3. | MaxAbsScaler | 0.972 |

## Выводы

Из полученных результатов в таблице 1 видно, что при увеличении размера обучающей выборки (train\_size) точность классификатора возрастает, достигая максимального значения при размере выборки 0.7 и выше.

Анализ результатов в таблице 2 показывает, что увеличение значения параметра n\_neighbors улучшает точность классификации до определенного момента (при n\_neighbors = 5, 9 и 15), после чего дальнейшее увеличение приводит к уменьшению точности.

Результаты из таблицы 3 показывают, что применение различных методов масштабирования данных не влияет на точность классификации. Это может быть обусловлено тем, что выбранные признаки (индексы 1 и 2) уже нормализованы и не требуют дополнительной предобработки.

ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

from sklearn import datasets

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler

def load\_data(train\_size=0.8):

wine = datasets.load\_wine()

data = wine.data[:, :2]

target = wine.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, target, train\_size=train\_size, random\_state=42)

return X\_train, X\_test, y\_train, y\_test

def train\_model(X\_train, y\_train, n\_neighbors=15, weights='uniform'):

model = KNeighborsClassifier(n\_neighbors=n\_neighbors, weights=weights)

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

return model

def predict(clf, X\_test):

predictions = clf.predict(X\_test)

return predictions

def estimate(res, y\_test):

accuracy = (res == y\_test).mean()

return round(accuracy, 3)

def scale(data, mode='standard'):

if mode == 'standard':

scaler = StandardScaler()

elif mode == 'minmax':

scaler = MinMaxScaler()

elif mode == 'maxabs':

scaler = MaxAbsScaler()

else:

return None

scaled\_data = scaler.fit\_transform(data)

return scaled\_data