**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. 3341 |  | Моисеева А.Е. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Цель работы – изучить возможности анализа данных с использованием классификатора из библиотеки *sklearn*. В частности, необходимо узнать, как обучать модели классификации данных и оценить качество классификации. Требуется реализовать код, который будет обрабатывать данные о винах.

## Задание

​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.

В отчёте приведите (чек-лист преподавателя):

* описание реализации 5и требуемых функций
* исследование работы классификатора, обученного на данных разного размера  
  + приведите точность работы классификаторов, обученных на данных от функции load\_data со значением аргумента train\_size из списка: 0.1, 0.3, 0.5, 0.7, 0.9
  + оформите результаты пункта выше в виде таблицы
  + объясните полученные результаты
* исследование работы классификатора, обученного с различными значениями *n\_neighbors*
  + приведите точность работы классификаторов, обученных со значением аргумента *n\_neighbors*из списка: 3, 5, 9, 15, 25
  + в качестве обучающих/тестовых данных для всех классификаторов возьмите результат *load\_data* с аргументами по умолчанию (учтите, что для достоверности результатов обучение и тестирование классификаторов должно проводиться на одних и тех же наборах)
  + оформите результаты в виде таблицы
  + объясните полученные результаты
* исследование работы классификатора с предобработанными данными
  + приведите точность работы классификаторов, обученных на данных предобработанных с помощью скейлеров из списка: StandardScaler, MinMaxScaler, MaxAbsScaler
  + в качестве обучающих/тестовых данных для всех классификаторов возьмите результат *load\_data* с аргументами по умолчанию - учтите, что для достоверности сравнения результатов классификации обучение должно проводиться на одних и тех же данных, поэтому предобработку следует производить **после**разделения на обучающую/тестовую выборку.
  + оформите результаты в виде таблицы
  + объясните полученные результаты

## Выполнение работы

Для получения исходных данных и последующего их анализа была использована библиотека *sklearn*.

Функции:

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

Описание: функция загружает набор данных о вине из библиотеки *sklearn* и разделяет его на обучающую и тестовую выборки.

Параметры: *train\_size* – доля данных, которая будет использоваться для обучения (по умолчанию 0.8).

Результат*: X\_train, x\_test* – двумерные массивы, содержащие данные для обучения и тестирования соответственно; *y\_train, y\_test* – одномерные массивы, содержащие метки классов для обучения и тестирования соответственно.

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

Описание: функция создает и обучает классификатор *KNeihborsClassifier* на предоставленных обучающих данных.

Параметры: *X\_train* – двумерный массив, содержащий обучающие данные; *y\_train* – одномерный массив, содержащий метки классов для обучения; *n\_neighbors* – количество ближайших соседей, используемых в классификаторе (по умолчанию 15); *weights* – схема взвешивания соседей (по умолчанию ‘uniform’).

Результат: возвращает обученную модель *KNeighborsClassifier.*

*predict(model, X\_test):*

Описание: функция выполняет предсказание классов для тестовых данных с использованием обученной модели.

Параметры: model – обученная модель KNeighborsClassifier; X\_test – двумерный массив, содержащий тестовые данные.

Результат: возвращает массив предсказанных меток классов.

*scale(data, mode=’standard’):*

Описание: функция выполняет масштабирование данных с использованием одного из трех скейлеров: *StandardScaler, MinMaxScaler, MaxAbsScaler.*

Параметры: data – двумерный массив, содержащий данные для масштабирования; mode – тип скейлера (по умолчанию – *‘standard’*, кроме того может быть *‘minmax’, ‘maxabs’*).

Результат: возвращает масштабированные данные.

Исследование работы классификатора, обученного на данных разного размера

|  |  |
| --- | --- |
| train\_size | accuracy |
| 0.1 | 0.379 |
| 0.3 | 0.8 |
| 0.5 | 0.843 |
| 0.7 | 0.815 |
| 0.9 | 0.722 |

Таблица 1. Результаты работы классификатора, обученного на выборке разного размера

Точность классификатора увеличивается с ростом объёма обучающей выборки. Однако, после какого-то порога видно, что модель начинает переобучаться, от чего точность падает. По итогу, самую высокую точность мы получаем при train\_size == 0.5.

Исследование работы классификатора, обученного с различными значениями n\_neighbors

|  |  |
| --- | --- |
| train\_size | accuracy |
| 3 | 0.861 |
| 5 | 0.833 |
| 9 | 0.861 |
| 15 | 0.861 |
| 25 | 0.833 |

Таблица 2. Результаты работы классификатора, обученного с различными значениями n\_neighbors

Изменение значений n\_neighbors практически не влияет на точность.

Исследование работы классификатора с предобработанными данными

|  |  |
| --- | --- |
| mode | accuracy |
| Without scale | 0.861 |
| Standard | 0.889 |
| Minmax | 0.806 |
| Maxabs | 0.75 |

Таблица 3. Результаты работы классификатора с предобработанными данными в различных режимах scale

Предобработка данных с использованием StandardScaler показала наилучшие результаты. Этот скейлер нормализует данные, устраняя смещения и масштабируя их, что помогает модели лучше улавливать закономерности в данных. MinMaxScaler и MaxAbsScaler также улучшают точность по сравнению с необработанными данными, но не дают таких же высоких результатов, как StandardScaler.

Разработанный код см. в приложении А.

## Выводы

Были изучены возможности анализа данных с использованием классификатора из библиотеки *sklearn*. В результате работы была реализована модель классификации методом k-ближайших соседей для анализа данных о винах. Модель показала высокую точность классификации, что свидетельствует о хорошем качестве работы классификатора.

# Приложение А Исходный код программы

Название файла: main.py

import numpy as np

from sklearn.datasets import load\_wine

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 = load\_wine()

X = wine.data[:, :2]

y = wine.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, 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(model, X\_test):

predictions = model.predict(X\_test)

return predictions

def estimate(predictions, y\_test):

accuracy = np.mean(predictions == y\_test)

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