**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Информационные технологии»**

Тема: Введение в анализ данных

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Самокрутов А.Р. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Цель работы заключается в изучении основ анализа данных с использованием библиотеки *sklearn* языка программирования *Python*, создание классификатора, его обучение и применение для классификации данных. Также необходимо реализовать программу, которая анализирует существующий ассортимент.

## 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.

## Выполнение работы

Описание функций:

1. *load\_data(train\_size=0.8):* загружает набор данных о вине из библиотеки sklearn в переменную wine и разбивает их на обучающую и тестовую части с использованием функции *sklearn.model\_selection.train\_test\_split()* с установленным параметром рандомизации.

2*. train\_model(train\_x, train\_y, n\_neighbors=15, wights=’uniform’):* принимает на вход обучающую выборку (train\_x и train\_y) и аргументы n\_neighbors и weights. Длаее она создает объект класса KNeighborsClassifier и загружает в него данные n\_neighbors и weights, после чего обучает его на данных train\_x и train\_y, полученных в функции выше. Функция возвращает экземпляр классификатора.

3. *predict(classifier, test\_x):* принимает на вход обученный классификатор и тренировочный набор данных (test\_x), после чего производит предсказание классов для тестовых данных test\_x с помощью метода predict(). В качестве результата возвращает предсказанные данные.

4. *estimate(res, test\_y):* принимает результаты классификации и истинные метки тестовых данных (test\_y), считает процент предсказанных результатов, совпавших с «правильными» в test\_y с помощью функции accuracy\_score(). В качестве результата возвращает полученное отношение, округленное до 0,001.

5. *scale(data, mode=’standard’):* функцияпринимает аргумент, содержащий данные, и аргумент mode - тип скейлера (допустимые значения: 'standard', 'minmax', 'maxabs', для других значений возвращается None, значение по умолчанию - 'standard'), обрабатывает данные соответствующим скейлером методом fit\_transform(). В качестве результата возвращаются полученные после обработки данные.

Исследование работы классификатора, обученного на данных разного размера:

Таблица 1 – Точность работы классификаторов в зависимости от аргумента train\_size

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

Наибольшая точность достигается при значении train\_size = 0.5. Это явление можно обосновать так: при меньших значениях модель не получает достаточно данных для выявления правдоподобных зависимостей и закономерностей, а при значениях, больших 0.5, модель избыточно подстраивается под входные данные и теряет способность к их обощению на новую информацию.

Исследование работы классификатора, обученного с различными значениями n\_neighbors:

Таблица 2 – точность работы классификаторов в зависимости от аргумента n\_neighbors

|  |  |
| --- | --- |
| n\_neighbors | Точность работы |
| 3 | 0.861 |
| 5 | 0.833 |
| 9 | 0.861 |
| 15 | 0.861 |
| 25 | 0.833 |

Наибольшая точность достигается при значениях n\_neighbors = 9 и 15. Слишком маленькое значение n\_neighbors приводит к недообучению модели, а при слишком большом значении модель теряет способность к выявлению закономерностей, т.к. обучена на этих данных избыточно.

Исследование работы классификатора с предобработанными данными:

Таблица 3 – точность работы классификаторов в зависимости от скейлеров

|  |  |
| --- | --- |
| Скейлер | Точность работы |
| Standard | 0.889 |
| MinMax | 0.806 |
| MaxAbs | 0.75 |

Заметим, что наибольшая точность достигается при использовании StandardScaler. Можно предположить, что StandardScaler масштабирует данные таким образом, что их среднее значение равно 0, а стандартное отклонение — 1, что может улучшить работу некоторых моделей машинного обучения.

Разработанный код см. в приложении А.

## Выводы

Были изучены основы анализа данных на языке программирования *Python* с применением библиотеки *sklearn*.

В результате работы была написана программа на языке программирования *Python*, которая проводит анализ существующего ассортимента, проверив возможности инструмента классификации данных для выделения различных классов.

# Приложение А Исходный код программы

Название файла: main.py

from sklearn.datasets import load\_wine

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler

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

wine = load\_wine()

x = wine.data[:, 0:2]

y = wine.target

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x, y,

train\_size=train\_size,

test\_size=1 - train\_size,

random\_state=42)

return train\_x, test\_x, train\_y, test\_y

def train\_model(train\_x, train\_y, n\_neighbors=15, weights='uniform'):

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

classifier.fit(train\_x, train\_y)

return classifier

def predict(classifier, test\_x):

prediction\_y = classifier.predict(test\_x)

return prediction\_y

def estimate(res, test\_y):

accuracy = accuracy\_score(test\_y, res)

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