**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. 3344 |  | Пачев Д.К. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

**Цель работы**

Познакомиться с анализом данных, развить навыки работы с библиотекой scikit-learn.

**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\_splitмодуля 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**

* load\_data(train\_size) - функция загружает встроенный датасет "вино" из библиотеки scikit-learn. Затем она извлекает только первые два признака (X) и целевую переменную (y) из этого датасета. После этого функция разделяет данные на обучающий и тестовый наборы, используя train\_test\_split. Параметр train\_size позволяет настраивать размер обучающего набора.
* train\_model(X\_train, y\_train, n\_neighbors, weights) - функция создает модель классификатора ближайших соседей (KNeighborsClassifier) из библиотеки scikit-learn. Она обучает модель на переданных обучающих данных (X\_train) и соответствующих метках классов (y\_train). Параметры n\_neighbors и weights позволяют настраивать количество соседей и веса для классификации.
* predict(clf, X\_test) - функция использует обученную модель (clf), чтобы предсказать метки классов для переданных тестовых данных (X\_test) с помощью метода predict
* estimate(res, y\_test) - функция оценивает точность предсказаний, сравнивая предсказанные метки классов (res) с фактическими метками классов (y\_test) с помощью метрики accuracy\_score.
* scale(data, mode) - функция выполняет масштабирование данных в соответствии с выбранным режимом. Она принимает данные (data) и режим масштабирования (mode), который может быть "standard" для стандартизации, "minmax" для мин-макс масштабирования или "maxabs" для масштабирования по максимальной абсолютной величине. Функция возвращает масштабированные данные.

Исследуем работу классификатора, обученного на данных разного размера

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

Из таблицы видно, что при малых значениях train\_size, точность классификатора мала, так как мало данных для тренировки модели. При значение train\_size = 0.5 достигается максимальная точность, но при последующем увеличении значения точность снижается, что говорит о том, что слишком много данных для обучения снизят эффективность, и приведут к переобучению модели, и увеличению времени обучения.

Исследуем работу классификатора, обученного с различными значениями n\_neighbors

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

Из таблицы видим, что точность работы при различных значениях n\_neighbors несильно различается. При значениях 3, 9 и 15 достигается максимальная точность, что означает, что для данного набора данных эти значения наиболее оптимальны.

Исследуем работу классификатора с предобработанными данными

|  |  |
| --- | --- |
| Скейлеры | Точность работы |
| StandardScaler | 0.417 |
| MinMaxScaler | 0.417 |
| MaxAbsScaler | 0.278 |

Из таблицы видим, что при использовании StandardScaler и MinMaxScaler точность работы не отличается, а при использовании MaxAbsScaler точность меньше, это говорит о том, что выбор способа масштабирования влияет на точность работы. При данном датасете лучшая точность достигается при StandardScaler и MinMaxScaler.

**Тестирование**

Результаты тестирования представлены в Таблице 1

Таблица 1 - Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1. | X\_train, X\_test, y\_train, y\_test = load\_data(0.5)  clf = train\_model(X\_train, y\_train)  res = predict(clf, X\_test)  est = estimate(res, y\_test)  print(est) | 0.843 | - |
| 2. | X\_train, X\_test, y\_train, y\_test = load\_data(0.5) standard\_scaled\_x = scale(X\_train) clf = train\_model(standard\_scaled\_x, y\_train) res = predict(clf, X\_test) est = estimate(res, y\_test) print(est) | 0.371 | - |

**Выводы**

В ходе выполнения лабораторной работы были получены навыки анализа данных на языке Python при помощи библиотеки scikit-learn.

# 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.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler

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

wine = datasets.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'):

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

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

return classifer

def predict(clf, X\_test):

return clf.predict(X\_test)

def estimate(res, y\_test):

return round(accuracy\_score(y\_true=y\_test, y\_pred=res), 3)

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

if mode == 'standard':

scaler = StandardScaler()

elif mode == 'minmax':

scaler = MinMaxScaler()

elif mode == 'maxabs':

scaler = MaxAbsScaler()

else: return None

return scaler.fit\_transform(data)