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 _____ Мурдасов М.К.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Введение в анализ данных. Изучение основных инструментов анализа данных на Python.

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, y_train, X_test, 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 с 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

1) *load_data()*:

   Загружает данные о вине из библиотеки sklearn и с помощью *train_test_split()* разделяет данные на тренировочные и тестируемые массивы. Аргумент *train_size* задает размер обучающей выборки и по умолчанию равен 0.8.

2) *train_model()*:

   Обучает и возвращает модель классификатора k-ближайштх соседей (*KNeighborsClassifier*) на выборке *X_train* с метками *y_train*. Аргументы *n_neighbors* задает кол-во соседей, а *weight* - весовую функцию для классификатора. По умолчанию *n_neighbors* равен 15 и *weights равен 'uniform'*.

3) *predict()*:

   Предсказывает значения меток, основываясь на тестовой выборке *X_test*.

4) *estimate():*

   Вычисляет точность классификации с помощью *accuracy_score*() из sklearn.*metrics* и возвращает ее значение, округленное до 0.001.

5) *scale():*

   Принимает массив данных X, режим масштабирования *mode* и возвращает масштабированный массив данных. Допустимые значения для *mode*: *'standard'*, *'minmax'*, *'maxabs'*. Если значение *mode* не является допустимым, функция возвращает *None*. Если *mode = 'standard'*, функция использует стандартное масштабирование (*StandardScaler*), если *mode = 'minmax'* - мини-максимальное масштабирование (*MinMaxScaler*), если *mode = 'maxabs'* - масштабирование по максимальному абсолютному значению (*MaxAbsScaler*). Масштабирование выполняется с помощью соответствующих классов из модуля *sklearn.preprocessing*.

Исследование работы классификатора, обученного на данных разного размера:

load_data с размерами данных	Точность работы классификатора
load_data(0.1)	0.379
load_data(0.3)	0.8
load_data(0.5)	0.843
load_data(0.7)	0.815
load_data(0.9)	0.722

Можно заметить, что точность классификации зависит от размера выборки. Слишком маленькая и слишком большая выборки снижают точность, тк имеют маленький объем данных или, напротив, слишком большой разброс этих данных.

Исследование работы классификатора, обученного с различными значениями *n_neighbors*:

значения *n_neighbors*	Точность работы классификатора
3	0.861
5	0.833
9	0.861
15	0.861
25	0.833

Можно заметить, что количество соседей незначительно влияет на точность работы классификатора.

Исследование работы классификатора с предобработанными данными:

Метод предобработки	Точность работы классификатора

StandardScaler	0.417
MinMaxScaler	0.417
MaxAbsScaler	0.278

Из полученных результатов можно заметить, что точность классификации зависит от способа масштабирования данных. При использовании стандартного масштабирования (StandardScaler) и минимакс-масштабирования (MinMaxScaler) точность классификации составляет 0.417, а при использовании максимального абсолютного масштабирования (MaxAbsScaler) точность классификации ниже и составляет 0.278. Таким образом, выбор способа масштабирования данных имеет влияние на точность классификации. В данном случае, стандартное масштабирование и минимакс-масштабирование показали лучшие результаты.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	X_train, X_test, y_train, y_test = load_data(0.3) scaled_x = scale(X_train) scaled_x_mm = scale(X_train, mode='minmax') scaled_x_abs = scale(X_train, mode='maxabs') c1 = train_model(scaled_x, y_train, 9) c3 = train_model(scaled_x_mm, y_train, 9) c5 = train_model(scaled_x_abs, y_train, 9) r1 = predict(c1, X_test) r3 = predict(c3, X_test) r5 = predict(c5, X_test) e1 = estimate(r1, y_test) e3 = estimate(r3, y_test) e5 = estimate(r5, y_test) print(e1, e3, e5)	0.368 0.392 0.384	Корректно

Выводы

Были изучены основные инструменты анализа данных в языке Python. Получены навыки работы с библиотекой sklearn.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Murdasov_Mikhail_lb3.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
    y = wine.target
    X_train, X_test, y_train, y_test = train_test_split(x[:, [0,1]], 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"):
    classifier = KNeighborsClassifier(n_neighbors = n_neighbors, weights = weights)
    classifier.fit(X_train, y_train)

    return classifier

def predict(classifier, X_test):
    pred = classifier.predict(X_test)

    return pred

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)
```