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 _____ Сербиновский Ю.М.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Введение в анализ данных. Ознакомление с базовыми концепциями и инструментами анализа данных на языке 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

Реализуйт 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 берет данные о вине из библиотеки sklearn, разделяет их на две части: обучающую и тестовую. Разделение происходит с помощью функции train_test_split из модуля sklearn.model_selection. Функция load_data возвращает четыре массива: X_train (обучающие данные), X_test (тестовые данные), y_train (метки для обучающих данных), y_test (метки для тестовых данных). По умолчанию 80% данных используются для обучения, а оставшиеся 20% - для тестирования (train_size=0.8).

Функция train_model обучает модель классификатора "K-ближайших соседей" (KNeighborsClassifier) на обучающей выборке X_train с использованием меток y_train. Классификатор использует алгоритм поиска k ближайших соседей для определения класса нового объекта. Функция принимает два параметра: n_neighbors - количество соседей, которые используются для классификации (по умолчанию 15), weights - весовая функция, которая определяет, как учитывать влияние соседей (по умолчанию 'uniform', то есть все соседи имеют одинаковый вес).

Функция predict принимает обученную модель классификатора и тестовые данные X_test. Она предсказывает метки для тестовых данных и возвращает массив res с предсказанными метками.

Функция estimate сравнивает предсказанные метки res с истинными метками y_test, используя функцию accuracy_score из модуля `sklearn.metrics`. Она рассчитывает точность классификации - долю правильно классифицированных объектов. Результат округляется до трех знаков после запятой.

Функция `scale` изменяет масштаб данных в массиве `X`. Она принимает режим масштабирования `mode`, который определяет метод преобразования данных: 'standard' - Стандартное масштабирование (StandardScaler) центрирует данные и приводит их к единичному стандартному отклонению; 'minmax' - мини-максимальное масштабирование (MinMaxScaler) преобразует данные в диапазон от 0 до 1; 'maxabs' - масштабирование по максимальному

абсолютному значению (MaxAbsScaler) делит каждый элемент на максимальное абсолютное значение в массиве. Если `mode` не является допустимым, функция возвращает `None`.

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

Эксперименты показывают, что точность классификации напрямую связана с размером обучающей выборки. При очень маленькой выборке (10% от всех данных) точность классификации оказывается низкой (37.9%). Это объясняется недостатком информации для обучения модели, которая не успевает "увидеть" все важные особенности данных. С увеличением размера выборки точность классификации растет и достигает пика при 50% данных (84.3%). Это означает, что модель получает достаточно информации для эффективного обучения. Однако, дальнейшее увеличение размера выборки не приводит к существенному повышению точности, а при 90% данных даже наблюдается ее снижение до 72.2%. Это говорит о том, что слишком большая выборка может быть неэффективной. Вероятно, модель начинает "переобучаться", то есть слишком сильно подстраивается под обучающую выборку, теряя способность обобщать знания на новые данные. Кроме того, увеличение выборки приводит к росту времени обучения модели. Таким образом, оптимальный размер выборки для классификации должен быть найден в балансе между достаточностью информации для обучения и риском переобучения.

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

Исследование влияния параметра n_neighbors на точность классификации показало, что различия в точности между моделями с разным количеством соседей не существенны. Наилучшие результаты (0.861) были получены при значениях n_neighbors, равных 3, 9 и 15. Модели с 5 и 25 соседями показали немного меньшую точность (0.833). Это говорит о том, что для данного набора данных оптимальным значением n_neighbors может быть 3, 9 или 15. Однако, поскольку разница в точности между этими вариантами незначительна, выбор конкретного значения может быть обусловлен другими факторами, например, желаемым компромиссом между точностью и скоростью работы модели.

Исследование работы классификатора с предобработанными данными:

Метод предобработки	Точность работы классификатора
StandardScaler	0.417
MinMaxScaler	0.417
MaxAbsScaler	0.278

Анализ точности классификации при разных способах масштабирования данных показал, что выбор метода масштабирования может значительно влиять на результат. Стандартное масштабирование (StandardScaler) и минимакс-масштабирование (MinMaxScaler) привели к одинаковой точности классификации (0.417). Использование максимального абсолютного

масштабирования (MaxAbsScaler) привело к значительно худшему результату (0.278). Таким образом, для данного набора данных стандартное масштабирование и минимакс-масштабирование показали себя более эффективными, чем максимальное абсолютное масштабирование. Важно помнить, что выбор оптимального метода масштабирования зависит от конкретного набора данных и задачи.

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(0.2) scaled_X_st = scale(X_train) scaled_X_mm = scale(X_train, 'minmax') scaled_X_ma = scale(X_train, 'maxabs') model_st = train_model(scaled_X_st, y_train) model_mm = train_model(scaled_X_mm, y_train) model_ma = train_model(scaled_X_ma, y_train) res_st = predict(model_st, X_test) res_mm = predict(model_mm, X_test) res_ma = predict(model_ma, X_test) print(estimate(res_st, y_test)) print(estimate(res_mm, y_test)) print(estimate(res_ma, y_test))	0.336 0.336 0.483	-

Выводы

Были получен опыт работы с базами данных. Были изучены базовые концепции и инструменты анализа данных на языке Python

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_train, X_test, y_train, y_test = train_test_split(wine.data[:, :2],
wine.target, train_size=train_size, random_state=42)
    return X_train, X_test, y_train, y_test
    pass

def train_model(X_train, y_train, n_neighbors=15, weights='uniform'):
    model = KNeighborsClassifier(n_neighbors=n_neighbors, weights=weights)
    model = model.fit(X_train, y_train)
    return model
    pass

def predict(clf, X_test):
    return clf.predict(X_test)
    pass

def estimate(res, y_test):
    return float('{:.3f}'.format(accuracy_score(res, y_test)))
    pass

def scale(X, mode='standard'):
    if mode not in ['standard', 'minmax', 'maxabs']:
        return None
    elif (mode == 'standard'):
```

```
    model = StandardScaler()
if mode == 'minmax':
    model = MinMaxScaler()
elif mode == 'maxabs':
    model = MaxAbsScaler()
model = model.fit_transform(X)
return model
```