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. 3344 _____ Ханнанов А.Ф.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Ознакомиться с библиотеками для анализа данных.

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. Реализация функций:
   1.1. load_data – функция загружает данные в переменную wine, после этого выбираются первые два столбца и записываются в x, а поля target в y. В конце данные разделяются и возвращаются
   1.2. train_model – создаётся классификатор и обучается на основе тестовых наборов, после этого он возвращается
   1.3. predict – используется обученный классификатор для прогнозирования меток тестовых данных и возвращает эти метки
   1.4. estimate – вычисляет точность прогнозов через сравнение предсказанных и заданными метками, возвращает точность
   1.5. scale – масштабирует данные и возвращает их

2. Обучение на данных разного размера (табл. 1):

Таблица 1 – Результаты работы классификатора

Размер набора	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Точность	0.379	0.797	0.8	0.822	0.843	0.819	0.815	0.861	0.722

Видно, что при увеличении аргумента точность увеличивается, но при значении 0.9 он уменьшается. Это связано с переобучением модели, она получает больше о “шуме”, а не о закономерностях.

3. Обучение с различными значениями n_neighbors (табл. 2):

Таблица 2 – результаты работы классификатора

Количество соседей	3	5	9	15	25
Точность	0.861	0.833	0.861	0.861	0.833

При увеличении количества соседей в среднем идёт рост точности, но при большом их количестве так же происходит переобучение модели, поэтому точность уменьшается.

4. Обучение с пред обработанными данными (табл. 3):

Таблица 3 – результаты работы классификатора

Скейлер	standart	minmax	maxabs
Точность	0.889	0.806	0.806

Особой разницы между скейлерами нет. В основном она зависит от особенности настроек конкретных данных.

Выводы

В ходе лабораторной работы прошло ознакомление с основами анализа данных. Был получен опыт анализа данных с помощью библиотек языка python.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Khannanov_Artem_lb3.py

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from    sklearn.preprocessing    import    MaxAbsScaler,    MinMaxScaler,
StandardScaler
import numpy as np


def load_data(train_size=0.8):
    wine = datasets.load_wine()
    x = wine.data[:, [0, 1]]
    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'):
          classifier   =   KNeighborsClassifier(n_neighbors=n_neighbors,
weights=weights)
    classifier.fit(X_train, y_train)

    return classifier


def predict(clf, X_test):
    prd = clf.predict(X_test)
    return prd


def estimate(res, y_test):
    correct = np.sum(res == y_test)
    total = len(y_test)

    return round(correct / total, 3)


def scale(args, mode='standard'):
    if mode == 'standard':
        scaler = StandardScaler()
    elif mode == 'minmax':
        scaler = MinMaxScaler()
    elif mode == 'maxabs':
        scaler = MaxAbsScaler()
    else:
        return None
    scaled = scaler.fit_transform(args)

    return scaled
```