

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. 3384

Рудаков А.Л.

Преподаватель

Иванов Д.В.

Санкт-Петербург

2024

Цель работы.

Изучить теорию об анализе данных в Python. Применить знания на практике реализовав программный код 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, 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 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.

В отчёте приведите (чек-лист преподавателя):

- ⑩ описание реализации 5и требуемых функций

- ⑩ исследование работы классификатора, обученного на данных разного размера
- ⑩ приведите точность работы классификаторов, обученных на данных от функции `load_data` со значением аргумента `train_size` из списка: 0.1, 0.3, 0.5, 0.7, 0.9
- ⑩ оформите результаты пункта выше в виде таблицы
- ⑩ объясните полученные результаты
- ⑩ исследование работы классификатора, обученного с различными значениями `n_neighbors`
- ⑩ приведите точность работы классификаторов, обученных со значением аргумента `n_neighbors` из списка: 3, 5, 9, 15, 25
- ⑩ в качестве обучающих/тестовых данных для всех классификаторов возьмите результат `load_data` с аргументами по умолчанию (учтите, что для достоверности результатов обучение и тестирование классификаторов должно проводиться на одних и тех же наборах)
- ⑩ оформите результаты в виде таблицы
- ⑩ объясните полученные результаты
- ⑩ исследование работы классификатора с предобработанными данными
- ⑩ приведите точность работы классификаторов, обученных на данных предобработанных с помощью скейлеров из списка: `StandardScaler`, `MinMaxScaler`, `MaxAbsScaler`
- ⑩ в качестве обучающих/тестовых данных для всех классификаторов возьмите результат `load_data` с аргументами по умолчанию - учтите, что для достоверности сравнения результатов классификации обучение должно проводиться на одних и тех же данных, поэтому предобработку следует производить **после** разделения на обучающую/тестовую выборку.
- ⑩ оформите результаты в виде таблицы

⑩ объясните полученные результаты

Выполнение работы.

Функция `load_data()` загружает данные при помощи `sklearn.datasets.load_wine()` и разбивает их для обучения и тестирования при помощи `train_test_split()`.

Функция `train_model()` обучает модель, создавая экземпляр классификатора `KNeighborsClassifier` и обучая его при помощи метода `sklearn.neighbors.KNeighborsClassifier.fit()`.

Функция `predict()` предсказывает классы у выборки при помощи метода `sklearn.neighbors.KNeighborsClassifier.predict()`.

Функция `estimate()` оценивает качество полученных результатов (выводит точность полученной модели) при помощи `sklearn.metrics.accuracy_score()`.

Функция `scale()` применяет скейлеры к поданным данным. Используемые скейлеры: `sklearn.preprocessing.StandardScaler()`, `sklearn.preprocessing.MinMaxScaler()`, `sklearn.preprocessing.MaxAbsScaler()`, после чего применяет скейлеры к данным при помощи `sklearn.preprocessing._Scaler().scaler.fit_transform()`.

Точность работы классификаторов, обученных на данных от функции `load_data` со значением аргумента `train_size`:

<code>train_size</code>	0.1	0.3	0.5	0.7	0.9
<code>est</code>	0.379	0.8	0.843	0.815	0.722

Полученные результаты объясняются тем, что если `train_size` мал, то модели недостаточно данных, чтобы быть точной, если же он наоборот велик, то модели не хватает данных на тесты, поэтому она также становится менее точной.

Точность работы классификаторов, обученных со значением аргумента $n_neighbors$:

n_neighbors	3	5	9	15	25
scaler	None	Standart	MinMax	MaxAbs	
est	0.861	0.833	0.861	0.861	0.833

Полученные результаты примерно равны, так как модели достаточно соседних значений для точности во всех случаях.

Точность работы классификаторов, обученных на данных предобработанных с помощью скейлеров:

scaler	None	Standart	MinMax	MaxAbs
scaler	None	Standart	MinMax	MaxAbs
est	0.861	0.798	0.809	0.82

Полученные результаты объясняются тем, что скейлеры «сглаживают данные» что дает погрешность при обучении.

Разработанный программный код см. в приложении А.

Выводы.

Была изучена теория об анализе данных в Python. Был создан программный код внутри которого были написаны функции, требуемые по заданию, внутри которых использовались элементы sklearn и numpy.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main_lb3.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 import preprocessing
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'):
    clf = KNeighborsClassifier(n_neighbors = n_neighbors, weights =
weights)
    clf.fit(X_train, y_train)
    return clf

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

def estimate(res, y_test):
    accuracy = accuracy_score(res, y_test)
    accuracy = np.round(accuracy, 3)
    return accuracy

def scale(data, mode = 'standard'):
    if mode == 'standard':
        scaler = preprocessing.StandardScaler()
    elif mode == 'maxabs':
        scaler = preprocessing.MaxAbsScaler()
    elif mode == 'minmax':
        scaler = preprocessing.MinMaxScaler()
    else:
        return None
    sc_data = scaler.fit_transform(data)
    return sc_data
```