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 _____ Волков А.А

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Познакомиться с анализом данных при помощи Python.

Изучить библиотеки, позволяющие работать с данными.

Рассмотреть возможность визуализации в виде таблиц, графиков, диаграмм.

Обучиться классификации и кластеризации данных.

Применить полученные знания при анализе датасета для решения поставленной задачи.

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.

Выполнение работы

1) Функция load_data():

В функции загружается датасет (load_wine()) и при помощи train_test_split происходит разделение данных на тренировочный и тестируемый наборы, которые являются возвращаемым значением. Аргумент train_size определяет размер обучающей выборки и по умолчанию равен 0.8.

2) Функция train_model():

В функции из полученных ранее тренировочных данных обучается модель классификатора (KNeighborsClassifier), возвращаемая в программу.

3) Функция predict():

В функции применяется метод predict для полученного ранее классификатора, в который передается набор тестируемых данных (X_test). Набор предсказываемых значений возвращается.

4) Функция estimate():

Функция вычисляет точность классификации с помощью функции accuracy_score модуля sklearn.metrics и возвращает ее округленную до трех знаков после запятой.

5) Функция scale():

Функция принимает массив данных X и режим масштабирования mode, возвращая масштабированный массив. Допустимые значения для mode: standard, minmax, maxabs. Если mode не допустим, функция возвращает None. В зависимости от значения mode, используются следующие классы из sklearn.preprocessing: standard - StandardScaler, minmax - MinMaxScaler, maxabs - MaxAbsScaler.

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

Анализируя полученные результаты, можно увидеть, что точность классификации существенно зависит от размера обучающей выборки. При маленькой выборке (0.1) точность составляет всего 0.379, что связано с недостаточным количеством данных для адекватного обучения модели. С увеличением размера выборки точность растет, достигая своего пика при размере 0.5, где точность составляет 0.843. Это указывает на то, что увеличение объема данных до определенного уровня способствует лучшему обучению модели. Однако дальнейшее увеличение размера выборки до 0.7 и 0.9 не приносит существенных улучшений, а в случае с выборкой 0.9 точность даже снижается до 0.722. Это снижение может быть связано с уменьшением объема тестовой выборки, что приводит к менее стабильной оценке модели. Таким образом, оптимальный размер выборки для обучения модели находится в диапазоне от 0.5 до 0.7, где достигается баланс между достаточным объемом данных для обучения и достаточным объемом данных для тестирования.

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

Анализируя данные по работе классификатора с различными значениями параметра n_neighbors, можно отметить, что точность классификации варьируется незначительно. Наибольшая точность, равная 0.861, достигается при значениях n_neighbors, равных 3, 9 и 15. При значениях n_neighbors, равных 5 и 25, точность немного ниже и составляет 0.833. Это показывает, что для данного набора данных оптимальными значениями параметра n_neighbors являются 3, 9 или 15. Однако разница в точности между этими значениями и значениями 5 и 25 достаточно мала, что указывает на стабильность модели при изменении числа соседей в указанном диапазоне.

Исследование работы классификатора с предобработанными данными:

Метод предобработки	Точность работы классификатора
StandardScaler	0.417
MinMaxScaler	0.417
MaxAbsScaler	0.278

Из анализа данных о работе классификатора с предобработанными данными видно, что точность классификации варьируется в зависимости от примененного метода масштабирования данных. При использовании стандартного и минимакс-масштабирования точность составляет 0.417, что указывает на схожий эффект обоих методов на модель. Однако, при использовании максимального абсолютного масштабирования точность снижается до 0.278. Таким образом, выбор метода масштабирования данных имеет значительное влияние на точность классификации. В данном случае, стандартное и минимакс-масштабирование показали схожие результаты, в то время как максимальное абсолютное масштабирование привело к снижению точности.

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(0.6) 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)	0.375 0.361 0.5	Программа работает корректно

Выводы

Ознакомление с основополагающими инструментами анализа данных.

Получен опыт работы с библиотеками Pandas, Matplotlib, Seaborn, scikit.

Были изучены возможности визуализации данных, доступные в вышеперечисленных библиотеках

Освоены идеи классификации и кластеризации набора данных.

Реализована программа, проводящая классификацию данных при помощи классификатора библиотеки scikit, оценены результаты ее работы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb_3.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[:, :2], 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):
    pred = clf.predict(X_test)
    return pred


def estimate(res, y_test):
    return round(accuracy_score(y_true=y_test, y_pred=res), 3)


def scale(X, mode='standard'):
    if mode not in ['standard', 'minmax', 'maxabs']:
```

```
        return None

    scaler = StandardScaler()
    if mode == 'minmax':
        scaler = MinMaxScaler()
    elif mode == 'maxabs':
        scaler = MaxAbsScaler()

    scaler = scaler.fit(X)
    x_scaled = scaler.transform(X)

    return x_scaled 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


# Загрузка данных
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[:, :2], y, train_size=train_size, random_state=42
    )
    return X_train, X_test, y_train, y_test


# Обучение модели
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


# Применение модели
def predict(clf, X_test):
```

```
    return clf.predict(X_test)


# Оценка качества результатов
def estimate(res, y_test):
    return round(accuracy_score(y_true=y_test, y_pred=res), 3)


# Предобработка данных
def scale(X, mode="standard"):
    if mode == "standard":
        scaler = StandardScaler()
    elif mode == "minmax":
        scaler = MinMaxScaler()
    elif mode == "maxabs":
        scaler = MaxAbsScaler()
    else:
        return None

    scaler.fit(X)
    return scaler.transform(X)
```