

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

Студентка гр. 3344

Коняева М.В.

Преподаватель

Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Целью работы является 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 `(в качестве состояния рандомизатора функции train_test_split sklearn.model_selection 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

Функция *load_data()* загружает набор данных о вине. Затем извлекает только первые два признака из данных. После этого данные разбиваются на обучающую и тестовую выборки с помощью функции *train_test_split* в соотношении, заданном параметром *train_size* (по умолчанию 80% данных используется для обучения). Наконец, функция возвращает четыре массива: *X_train*, *X_test*, *y_train*, *y_test*.

Функция *train_model()* создает и обучает модель классификации методом k-ближайших соседей (KNN). Она принимает в качестве входных параметров обучающие данные для признаков (*X_train*) и соответствующие метки (*y_train*). Параметры *n_neighbors* и *weights* используются для определения числа соседей и весовых коэффициентов соответственно. Затем функция создает экземпляр классификатора *KNeighborsClassifier* с заданными параметрами и обучает его на предоставленных обучающих данных. В конце функция возвращает обученную модель.

Функция *predict()* принимает обученную модель классификатора и тестовую выборку *X_test*, предсказывает метки для этой выборки и возвращает массив предсказанных меток.

Функция *estimate()* принимает предсказанные значения *res* и истинные метки *y_test* в качестве входных данных. Она использует функцию *accuracy_score* для оценки точности предсказаний модели. Затем округляет полученное значение точности до трех знаков после запятой. В итоге функция возвращает оценку точности модели.

Функция *scale()* принимает данные и режим масштабирования *mode* в качестве входных параметров. В зависимости от выбранного режима масштабирования (стандартное, минимаксное или максимальное абсолютное значение), функция создает соответствующий экземпляр соответствующего скейлера (*StandardScaler*, *MinMaxScaler* или *MaxAbsScaler*). Затем она применяет выбранный скейлер к данным и возвращает масштабированные данные. Если указан неверный режим масштабирования, функция вернет *None*.

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

Таблица 1 – Результаты работы классификатора

Размер набора	0.1	0.3	0.5	0.7	0.9
Точность	0.379	0.8	0.843	0.815	0.722

Результаты показывают, что существует оптимальный размер обучающей выборки, при котором модель демонстрирует наилучшую производительность на тестовых данных, и дальнейшее увеличение размера выборки может привести к ухудшению результатов из-за переобучения.

2. 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

Таким образом, результаты указывают на то, что модель KNN достигает высокой точности на тестовых данных при различном количестве соседей, и выбор оптимального числа соседей в диапазоне от 3 до 15 не существенно влияет на результаты.

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

Таблица 3 – результаты работы классификатора

Скейлер	Standard	MinMax	MaxAbs
Точность	0.417	0.417	0.278

Из полученных данных следует, что точность классификации различается в зависимости от способа масштабирования данных. В данном случае, стандартное (*StandardScaler*) и минимакс-масштабирование (*MinMaxScaler*) показали более высокую точность.

Разработанный программный код см. в приложении А. Результаты тестирования см. в приложении Б.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Тест	Выходные данные	Комментарии
1.	<pre>X_train, X_test, y_train, y_test = load_data() 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, 3) c3 = train_model(scaled_x_mm, y_train, 3) c5 = train_model(scaled_x_abs, y_train, 3) 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)</pre>	0.389 0.389 0.528	Данные обработаны корректно

Выводы

В результате выполнения лабораторной работы были приобретены знания о базовых понятиях и инструментах анализа данных на языке Python. Проведенное знакомство с основами анализа данных позволило получить опыт работы с соответствующими библиотеками Python (*Scikit-learn*).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.py

```
from sklearn import datasets

def load_data(train_size=0.8):
    wine = datasets.load_wine()
    X = wine.data[:, :2]
    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'):
    knn = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors,
weights=weights)
    knn.fit(X_train, y_train)
    return knn

def predict(clf, X_test):
    y_pred = clf.predict(X_test)
    return y_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

    scaled_data = scaler.fit_transform(data)
    return scaled_data
```