**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 _____ Якимова Ю.А.

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**

1. Описание реализации пяти требуемых функций:

Функция *load_data()* загружает набор данных о вине из библиотеки *sklearn* в переменную wine. После этого данные разбиваются на обучающую и тестовую выборки с помощью функции *train_test_split*. Функция возвращает четыре массива: *X_train, X_test, y_train, y_test*.

Функция *train_model()* создает и обучает модель классификации методом k-ближайших соседей. Параметры *n_neighbors* и *weights* используются для определения числа соседей и весовых коэффициентов соответственно. Затем функция создает экземпляр классификатора *KNeighborsClassifier* и обучает его на обучающей выборке. Функция возвращает обученную модель.

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

Функция *estimate()* принимает массив предсказанных меток *res* и массив истинных меток *y_test* в качестве входных данных. Вычисляет точность прогнозов через сравнение предсказанных с заданными метками. Функция возвращает оценку точности модели.

Функция *scale()* принимает данные и режим масштабирования *mode* в качестве входных параметров. В зависимости от выбранного режима масштабирования функция создает экземпляр соответствующего скейлера. Функция возвращает масштабированные данные.

2. Исследование работы классификатора, обученного на данных разного размера:

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 эффективность на тестовых данных, и дальнейшее увеличение размера выборки может привести к ухудшению результатов из-за переобучения.

3. Исследование работы классификатора, обученного с различными значениями *n_neighbors:*

Таблица 2 – результаты работы классификатора

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

Точность работы классификаторов с разными значениями n_neighbors почти не различаются, и выбор оптимального числа соседей в диапазоне от 3 до 15 не существенно влияет на результаты.

4. Исследование работы классификатора с предобработанными данными:

Таблица 3 – результаты работы классификатора

| Скейлер | Standard | MinMax | MaxAbs |
|---|---|---|---|
| Точность | 0.417 | 0.417 | 0.278 |

Видно, 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.

**Тестирование**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

| № п/п | Тест | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | `X_train, X_test, y_train, y_test = load_data()`<br>`scaled_x = scale(X_train)`<br>`scaled_x_mm = scale(X_train, mode='minmax')`<br>`scaled_x_abs = scale(X_train, mode='maxabs')`<br><br>`c1 = train_model(scaled_x, y_train, 3)`<br>`c3 = train_model(scaled_x_mm, y_train, 3)`<br>`c5 = train_model(scaled_x_abs, y_train, 3)`<br><br>`r1 = predict(c1, X_test)`<br>`r3 = predict(c3, X_test)`<br>`r5 = predict(c5, X_test)`<br><br>`e1 = estimate(r1, y_test)`<br>`e3 = estimate(r3, y_test)`<br>`e5 = estimate(r5, y_test)`<br>`print(e1, e3, e5)` | `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: Yakimova_Yuliya_lb3.py

```
from sklearn import datasets

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

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

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