**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* c 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 |
|  | 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) | 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