**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. Вариант 1

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Поддубный В. А. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Научиться работать с библиотекой scikit-learn, понять, для чего она используется, как обрабатывать входные данные, как классифицировать данные, методы классификации данных, как сравнить полученные результаты.

## 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, y\_train, X\_test, 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

Были реализованы следующие функции:

1. **load\_data(train\_ratio=0.8, random\_seed=42)**:
   * Загружает набор данных о вине из библиотеки sklearn.
   * Разбивает данные на обучающую и тестовую выборки в соответствии с заданным соотношением train\_ratio.
   * Возвращает X\_train, X\_test, y\_train, y\_test - обучающие и тестовые данные для признаков и целевой переменной, соответственно.
2. **train\_model(X\_train, y\_train, k\_neighbors=15, weight\_method='uniform')**:
   * Создает экземпляр классификатора KNeighborsClassifier с заданными параметрами.
   * Обучает модель на предоставленных данных X\_train и y\_train.
   * Возвращает обученный классификатор.
3. **predict(classifier, X\_test)**:
   * Принимает обученный классификатор и тестовые данные X\_test.
   * Выполняет классификацию данных и возвращает предсказанные метки классов.
4. **estimate(predicted\_labels, y\_test)**:
   * Сравнивает предсказанные метки классов с истинными метками y\_test.
   * Вычисляет и возвращает точность классификации (долю правильных ответов).
5. **scale(data, mode='standard')**:
   * Принимает данные и тип скейлера (standard, minmax, maxabs).
   * Выполняет предобработку данных с использованием выбранного скейлера.
   * Возвращает преобразованные данные.

Исследование работы классификатора

Влияние размера обучающей выборки

|  |  |
| --- | --- |
| **Значение train\_size** | **Точность работы классификатора** |
| 0.1 | 0.667 |
| 0.3 | 0.741 |
| 0.5 | 0.778 |
| 0.7 | 0.778 |
| 0.9 | 0.833 |

**Анализ результатов:**

* С увеличением размера обучающей выборки точность классификации в целом возрастает.
* При очень маленьком размере выборки (0.1) модель не получает достаточно информации для обучения, что приводит к низкой точности.
* При слишком большом размере выборки (0.9) возможно переобучение, когда модель слишком хорошо запоминает обучающие данные и плохо обобщает на новых данных.

Влияние количества соседей (n\_neighbors)

|  |  |
| --- | --- |
| **Значение n\_neighbors** | **Точность работы классификатора** |
| 3 | 0.861 |
| 5 | 0.833 |
| 9 | 0.889 |
| 15 | 0.861 |
| 25 | 0.806 |

**Анализ результатов:**

* Значение n\_neighbors оказывает существенное влияние на точность классификации.
* При небольшом количестве соседей (3, 5) модель может быть чувствительна к шуму в данных.
* При слишком большом количестве соседей (25) границы между классами могут размываться, что снижает точность.
* Оптимальное значение n\_neighbors зависит от конкретного набора данных.

Влияние предобработки данных

|  |  |
| --- | --- |
| **Тип скейлера** | **Точность работы классификатора** |
| StandardScaler | 0.889 |
| MinMaxScaler | 0.806 |
| MaxAbsScaler | 0.750 |

**Анализ результатов:**

* Предобработка данных с использованием StandardScaler привела к наилучшей точности классификации.
* Это связано с тем, что StandardScaler приводит данные к стандартному нормальному распределению, что может улучшить работу алгоритма k-ближайших соседей.
* MinMaxScaler и MaxAbsScaler показали менее высокую точность, возможно, из-за чувствительности к выбросам в данных.

## Выводы

Была написана программа, которая состоит из функции загрузки данных о винах, которая разделяет их на обучающие и тестовые данные, функции обучении модели, которая загружает в классификатор ближайших соседей обучающие данные, функция, которая применяет эту модель на тестовых данных, функция оценки результатов и их предобработка.

# Приложение А Исходный код программы

Название файла: main.py

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler

def load\_data(train\_ratio=0.8, random\_seed=42):

wine\_dataset = datasets.load\_wine()

features = wine\_dataset.data[:, :2]

target = wine\_dataset.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

features, target, train\_size=train\_ratio, random\_state=random\_seed)

return X\_train, X\_test, y\_train, y\_test

def train\_model(X\_train, y\_train, k\_neighbors=15, weight\_method='uniform'):

knn\_classifier = KNeighborsClassifier(n\_neighbors=k\_neighbors, weights=weight\_method)

knn\_classifier.fit(X\_train, y\_train)

return knn\_classifier

def predict(classifier, X\_test):

return classifier.predict(X\_test)

def estimate(predicted\_labels, y\_test):

return round((predicted\_labels == y\_test).mean(), 3)

def scale(data, mode='standard'):

scalers = {

'standard': StandardScaler(),

'minmax': MinMaxScaler(),

'maxabs': MaxAbsScaler()

}

selected\_scaler = scalers.get(mode)

return selected\_scaler.fit\_transform(data) if selected\_scaler else None