**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. 3343 |  | Иванов П.Д. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Изучить библиотеку scikit-learn и примененить классификатор K-ближайших соседей (KNN) для анализа данных. Исследовать влияние различных параметров модели и методов предобработки данных на точность классификации.

## 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 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 5 функций. Описание каждой функции и логика работы описаны ниже.

1. *load\_data()* - используется для загрузки данных.

* Загружает данные о вине из библиотеки sklearn.
* Извлекает только первые два столбца из данных и метки классов.
* Делит данные на обучающую и тестовую выборки в соответствии с параметром train\_size.
* Возвращает x\_train, x\_test, y\_train, y\_test.

1. *train\_model()* - используется для обучения модели.

* Создаёт экземпляр классификатора K-ближайших соседей с заданными параметрами n\_neighbors и weights.
* Обучает модель на данных x\_train и y\_train.
* Возвращает обученную модель.

1. *predict()* - используется для предсказания следующего значения.

* Выполняет предсказание меток классов для тестовых данных x\_test с использованием обученной модели model.
* Возвращает предсказанные значения y\_pred.

1. *estimate()* - используется для оценки качества работы модели.

* Вычисляет точность предсказаний как долю правильных предсказаний среди всех тестовых данных.
* Возвращает точность, округленную до трёх знаков после запятой.

1. *scale()* - используется для обработки входных данных по скейлеру.

* Масштабирует данные с использованием заданного скейлера (standard, minmax, maxabs).
* Возвращает масштабированные данные.
* Если передан некорректный режим, возвращает None.

Для исследования точности классификатора при различных размерах обучающей выборки использовались значения train\_size из списка: 0.1, 0.3, 0.5, 0.7, 0.9. Результаты приведены в таблице:

|  |  |
| --- | --- |
| train\_size | accuracy |
| 0.1 | 0.611 |
| 0.3 | 0.593 |
| 0.5 | 0.685 |
| 0.7 | 0.741 |
| 0.9 | 0.778 |

Из таблицы видно, что с увеличением размера обучающей выборки train\_size точность модели K-ближайших соседей возрастает.  
Это связано с тем, что больший объём обучающих данных позволяет модели лучше обучаться и делать более точные предсказания на тестовых данных.  
При малом размере обучающей выборки модель не получает достаточно информации для точного классифицирования новых данных.

Для исследования точности классификатора при различных значениях n\_neighbors использовались значения из списка: 3, 5, 9, 15, 25. Результаты приведены в таблице:

|  |  |
| --- | --- |
| n\_neighbors | accuracy |
| 3 | 0.741 |
| 5 | 0.759 |
| 9 | 0.759 |
| 15 | 0.759 |
| 25 | 0.741 |

Отсюда понятно, что точность классификатора незначительно меняется при различных значениях n\_neighbors. Оптимальное значение n\_neighbors для данного набора данных находится в диапазоне от 5 до 15. При слишком малом значении n\_neighbors модель становится более чувствительной к шуму в данных, а при слишком большом значении — модель может стать слишком сглаженной и терять точность.

Для исследования точности классификатора при использовании различных скейлеров использовались скейлеры из списка: StandardScaler, MinMaxScaler, MaxAbsScaler. Результаты приведены в таблице:

|  |  |
| --- | --- |
| scaler | accuracy |
| StandardScaler | 0.832 |
| MinMaxScaler | 0.801 |
| MaxAbsScaler | 0.779 |

Масштабирование данных улучшает качество классификации. Масштабирование данных помогает нормализовать их, что улучшает работу алгоритмов, таких как K-ближайших соседей.

Разработанный программный код см. в приложении А.

## Выводы

В результате выполнения работы был написан код для анализа данных о вине с использованием классификатора K-ближайших соседей (KNN) и библиотеки scikit-learn. Были реализованы функции для загрузки и предобработки данных, обучения и оценки модели. Также было проведено исследование влияния размера обучающей выборки, значения гиперпараметра n\_neighbors и различных методов предобработки данных на точность классификации.

# Приложение А Исходный код программы

Название файла: main.py

from sklearn.datasets import load\_wine  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler  
  
  
def load\_data(train\_size=0.8):  
 wine = 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 = KNeighborsClassifier(n\_neighbors=n\_neighbors, weights=weights)  
 knn.fit(X\_train, y\_train)  
 return knn  
  
  
def predict(model, X\_test):  
 y\_pred = model.predict(X\_test)  
 return y\_pred  
  
  
def estimate(y\_pred, y\_test):  
 accuracy = (y\_pred == y\_test).mean()  
 return round(accuracy, 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