**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 |  | Коршков А.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

Чтобы получить данные о винах, необходимо импортировать датасеты и вызвать функцию data\_wine. Чтобы взять данные для осей X и Y используются data и target соответственно. После чего делим данные на тренировочные и тестовые с помощью функции train\_test\_split в соответствии с заданным train\_size (в процентном соотношении, по умолчанию 0.8, т.е 80%). Чтобы взять первые 2 столбца что в тестовых, что в обучающих данных необходимо сделать двойной срез вида [ : , :2].

Чтобы исследовать работу классификатора необходимо подготовить вот такой код:

if \_\_name\_\_ == '\_\_main\_\_':

X\_train, X\_test, y\_train, y\_test = load\_data()

clf = train\_model(X\_train, y\_train)

res = predict(clf, X\_test)

est = estimate(res, y\_test)

print(est)

После получения тестовых данных необходимо создать модель классификации K-ближайших соседей, и с помощью функции predict выполнить классификацию тестовых данных и получить какой-то ответ. А с помощью функции estimate сделать оценку предсказанных данных (сравнить данные на соответствие с y\_test).

|  |  |
| --- | --- |
| Значение train\_size | Точность работы классификатора |
| 0.1 | 0,379 |
| 0.3 | 0,8 |
| 0.5 | 0,843 |
| 0.7 | 0,815 |
| 0.8 (default) | 0,861 |
| 0.9 | 0,722 |

Можно заметить, что при дефолтном значении train\_size = 0.8 достигается самая высокая точность. В принципе, можно заметить, что значение держится в среднем в районе 0.8 при train\_size >= 0.3 и train\_size <= 0.8. При значении 0.1 точность работы очень низкая, модели не хватает данных, чтобы обучиться и делать правильную оценку. При повышении значения train\_size выше 0.8 точность работы также начинает падать, т.к. модель начинает уже не предсказывать данные, а запоминать их (происходит переобучение). Она начинает терять способность обобщать на новые, ранее не виданные данные.

Рассмотрим, как значение n\_neighbours влияет на точность алгоритма ближайших соседей. Классификация вычисляется простым большинством голосов ближайших соседей каждой точки: точке запроса назначается класс данных, который имеет наибольшее количество представителей среди ближайших соседей точки. Количество этих соседей, с которыми нужно сравнить точку надо задать самостоятельно. Изменение данного значения может приводить к тому, что точка может изменить свою группу принадлежности.

|  |  |
| --- | --- |
| Значение n\_neighbors при train\_size=0.8 | Точность работы классификатора |
| 3 | 0,861 |
| 5 | 0,833 |
| 9 | 0,861 |
| 15 (default) | 0,861 |
| 25 | 0,833 |

В данном примере видно, что значение n\_neighbors не сильно влияет на точность работы. Видно, что точность незначительно уменьшается при значениях 5 и 25. При сильно больших значениях, точность будет уменьшаться уже к 0.7. Происходит это потому, что точки начинают принадлежать не к той группе, из-за того, что кол-во исследуемых соседей становиться больше, некоторые группы данных в окрестности исследуемой точки становятся больше и их влияние на исследуемую точку возрастает.

Для проверки точности работы скейлеров необходимо выполнить предобработку данных:

if \_\_name\_\_ == '\_\_main\_\_':

X\_train, X\_test, y\_train, y\_test = load\_data()

X\_train = scale(X\_train)

X\_test = scale(X\_test)

clf = train\_model(X\_train, y\_train)

res = predict(clf, X\_test)

est = estimate(res, y\_test)

print(est)

Стандартизация данных сводиться к тому, чтобы преобразовать данные к единому формату и представлению, который будет удобен для определённого вида обработки. Делать стандартизацию можно разными методами. Если говорить про стандартную стандартизацию, то она изменяет масштаб данных, так что среднее значение становится равным 0, а стандартное отклонение становится равным 1. Из каждого значения вычитается среднее из набора, а затем разделено на стандартное отклонение всего набора данных.

Скейлер MinMaxScaler преобразует значения по такой формуле:

Данные значения масштабируются так, чтобы значения находилась в диапазоне от 0 до 1.

Скейлер MaxAbsScaler масштабирует значения в диапазон от -1 до 1 с помощью деления каждого значения на максимальное значение по модулю.

|  |  |
| --- | --- |
| Типы скейлеров | Точность работы классификатора |
| StandardScaler | 0,889 |
| MinMaxScaler | 0,806 |
| MaxAbsScaler | 0,75 |

В данном случае видно, что скейлер по умолчанию отработал лучше всего и показал наивысшую точность, MaxAbsScaler показал худшие результаты среди представленных скейлеров.

Такое могло произойти, потому что MinMaxScaler и MaxAbsScaler чувствительны к выбросам, в то время как StandardScaler игнорирует выбросы.

## 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.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler

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, random\_state=42, train\_size=train\_size)

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

def train\_model(X\_train, y\_train, n\_neighbors=15, weights='uniform'):

return KNeighborsClassifier(n\_neighbors=n\_neighbors, weights=weights).fit(X\_train, y\_train)

def predict(clf, X\_test):

return clf.predict(X\_test)

def estimate(res, y\_test):

accuracy = accuracy\_score(y\_test, res)

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

return scaler.fit\_transform(data)

# Приложение Б ТЕСТИРОВАНИЕ

Тест №1

**Ввод (программа):**

if \_\_name\_\_ == '\_\_main\_\_':

X\_train, X\_test, y\_train, y\_test = load\_data()

print(X\_train[:3], X\_test[:3], sep="\n")

X\_train = scale(X\_train)

X\_test = scale(X\_test)

print(X\_train[:3], X\_test[:3], sep="\n")

**Вывод:**

[[14.34 1.68]

[12.53 5.51]]

[[13.64 3.1 ]

[14.21 4.04]]

[[ 1.66529275 -0.60840587]

[-0.54952506 2.7515415 ]]

[[0.71755938 0.91772503]

[1.4552507 1.86541545]]

**Примечание:**

Данный тест показывает, что применилась стандартизация по умолчанию, значения приведены к среднему значению 0 и среднему отклонению 1. \

Тест №2

**Ввод (программа):**

if \_\_name\_\_ == '\_\_main\_\_':

X\_train, X\_test, y\_train, y\_test = load\_data()

X\_train = scale(X\_train)

X\_test = scale(X\_test)

clf = train\_model(X\_train, y\_train)

res = predict(clf, X\_test)

est = estimate(res, y\_test)

print(est)

**Вывод:**

0.889

**Примечание:**

Данный тест показывает, что при стандартных параметрах разделения данных, стандартизации по умолчанию, можно добиться наивысшей точности.