**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. 3341 |  | Шаповаленко Е.В. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Целью работы является изучение основ анализа данных и написание программы на языке Python, анализирующей и классифицирующей данные с помощью библиотеки *sklearn*.

## 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.

В отчёте приведите (чек-лист преподавателя):

* описание реализации 5и требуемых функций
* исследование работы классификатора, обученного на данных разного размера  
  + приведите точность работы классификаторов, обученных на данных от функции load\_data со значением аргумента train\_size из списка: 0.1, 0.3, 0.5, 0.7, 0.9
  + оформите результаты пункта выше в виде таблицы
  + объясните полученные результаты
* исследование работы классификатора, обученного с различными значениями *n\_neighbors*
  + приведите точность работы классификаторов, обученных со значением аргумента *n\_neighbors*из списка: 3, 5, 9, 15, 25
  + в качестве обучающих/тестовых данных для всех классификаторов возьмите результат *load\_data* с аргументами по умолчанию (учтите, что для достоверности результатов обучение и тестирование классификаторов должно проводиться на одних и тех же наборах)
  + оформите результаты в виде таблицы
  + объясните полученные результаты
* исследование работы классификатора с предобработанными данными
  + приведите точность работы классификаторов, обученных на данных предобработанных с помощью скейлеров из списка: StandardScaler, MinMaxScaler, MaxAbsScaler
  + в качестве обучающих/тестовых данных для всех классификаторов возьмите результат *load\_data* с аргументами по умолчанию - учтите, что для достоверности сравнения результатов классификации обучение должно проводиться на одних и тех же данных, поэтому предобработку следует производить **после**разделения на обучающую/тестовую выборку.
  + оформите результаты в виде таблицы
  + объясните полученные результаты

## Выполнение работы

При написании программы были реализованы следующие функции:

*load\_data()* принимает на вход размер обучающей выборки *train\_size* (по умолчанию *0.8*). функция загружает данные о вине из библиотеки *sklearn* в переменную *wine* с помощью *load\_wine()*. Затем с помощью *train\_test\_split()* данные разбиваются на тренировочную и тестовую выборки.

*train\_model()* создаёт экземпляр классификатора, обученного *методом k-ближайших соседей* с помощью *KNeighborsClassifier()*. Количество соседей *n\_neighbors* по умолчанию равно *15*, веса – *‘uniform’*. Затем происходит обучение на переданных данных, а затем возвращается экземпляр классификатора.

*predict()* принимает обученный классификатор *cls* и с помощью его метода *predict()* выполняет классификацию из набора данных *X\_test*. Функция возвращает предсказанные данные.

*estimate()* принимает результаты классификации *res* и истинные метки тестовых данных *y\_test* и оценивает качество результатов классификации с помощью *accuracy\_score()*. Функция возвращает округленный до *0.001* результат.

*scale()* принимает на вход данные *data* и тип скейлера *mode* (по умолчанию *‘standard’*). Функция обрабатывает данные по одному из трёх скейлеров: ‘*standard’, ‘minmax’* и *‘maxabs’* и возвращает обработанные данные.

Точность работы классификаторов, обученных на разных размерах обучающей выборки *train\_size* для функции *load\_data()* представлены в табл. 1.

Таблица 1 – Точность работы классификаторов при разных train\_size

|  |  |
| --- | --- |
| Значение *train\_size* | Точность работы |
| 0.1 | 0.379 |
| 0.3 | 0.8 |
| 0.5 | 0.843 |
| 0.7 | 0.815 |
| 0.9 | 0.722 |

Как видно, наилучшая точность работы достигается при значениях *train\_size 0.5*; при меньших значениях модели не хватает данных для обучения, а при больших может происходить переобучение модели или недостаток данных для валидации прогресса при обучении.

Точность работы классификаторов, обученных с разными значениями n\_neighbors, но на одинаковых тренировочных данных, приведена в табл.2.

Таблица 2 – Точность работы при различных n\_neighbors

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

Видно, что точность работы меняется при различных значениях *n\_neighbors*. Наилучшие значения достигаются при значениях *n\_neighbors* 3, 9 и 15, хотя при *n\_heighbors*=3 точность работы, скорее, случайна, т.к. модель может быть подвержена шуму. При больших значениях *n\_neighbors* модель начинает учитывать слишком много данных, часть которых становится менее полезной.

Точность работы классификаторов, обученных на предобработанных данных с помощью различных скейлеров, приведена в табл. 3.

Таблица 3 – Точность работы при предобработанных данных

|  |  |
| --- | --- |
| Скейлер | Точность работы |
| *‘standard’* | 0.889 |
| *‘minmax’* | 0.806 |
| *‘maxabs’* | 0.75 |

Видно, что наилучшие результаты достигаются при скейлере *‘standard’*. Этот скейлер хорошо подходит для большинства алгоритмов машинного обучения.

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 4.

Таблица 4 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 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) | 0.861 | Стандартное обучение |
|  | X\_train, X\_test, y\_train, y\_test = load\_data()  X\_train\_scaled = scale(X\_train, 'minmax')  X\_test\_scaled = scale(X\_test, 'minmax')  clf = train\_model(X\_train\_scaled, y\_train)  res = predict(clf, X\_test\_scaled)  est = estimate(res, y\_test)  print(est) | 0.806 | Обучение со скейлером |

## Выводы

В ходе выполнения работы были изучены основы анализа данных на языке Python с применением библиотеки *sklearn*. Разработаны функции для выгрузки данных, обучения модели, применения модели, оценки её эффективности и предобработки данных. Была проанализирована точность работы моделей при различных условиях обучения.

# 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.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler

def load\_data(train\_size=0.8):

wine=load\_wine()

X=wine.data[:, 0:2]

y=wine.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=train\_size, test\_size=1-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'):

classifier = KNeighborsClassifier(n\_neighbors=n\_neighbors, weights=weights)

classifier.fit(X\_train, y\_train)

return classifier

def predict(clf, X\_test):

y\_pred = clf.predict(X\_test)

return y\_pred

def estimate(res, y\_test):

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

accuracy = round(accuracy, 3)

return accuracy

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