

**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

Бойцов В.А.

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` с 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(train\_size=0.8)* 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(X\_train, y\_train, n\_neighbors=15, weights='uniform')* 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(clf, X\_test)* принимает обученную модель и с помощью метода классификатора *predict()* выполняет классификацию из набора данных *X\_test*. Функция возвращает предсказанные данные.

*estimate(res, y\_test)* 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(data, mode='standard')* 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 <i>train_size</i>	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 или 0.7; 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  $n\_neighbors=3$  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. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<pre>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)</pre>	0.861	Стандартное обучение
2.	<pre>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, y_train) res = predict(clf, X_test) est = estimate(res, y_test) print(est)</pre>	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, оценки её эффективности и др. 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
```