**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 |  | Синицкая Д.В. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Изучение введения в анализ данных, создание программы на языке программирования Python, в которой реализуется работа с функциями библиотеки sklearn.

## Задание

Вариант 1. 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

1) Описание реализации пяти требуемых функций.

def load\_data(train\_size = 0.8) – функция загружает данные о вине из встроенного набора данных и разделяет их на обучающий и тестовые наборы. Принимает аргумент train\_size, который указывает долю данных, используемую для обучения. Возвращает обучающий набор признаков, тестовый набор признаков, метки классов обучающего набора и метки классов тестового набора.

def train\_model(X\_train, y\_train, n\_neighbors = 15, weights = 'uniform') – функция обучает модель классификации методом k-ближайших соседей. Принимает обучающий набор признаков, метки классов, число соседей и тип весов. Возвращает обученную модель с алгоритмом k-ближайших соседей.

def predict(clf, X\_test) – функция предсказывает метки классов на тестовых данных с использованием обученной модели. Принимает обученную модель и тестовый набор признаков. Возвращает предсказанные метки классов для тестового набора.

def estimate(res, y\_test) – функция оценивает точность модели по предсказанным результатам и истинным меткам классов с помощью метрики. Возвращает значение точности модели, округленное до трех знаков после запятой.

def scale(data, mode = 'standard') – функция масштабирует данные с использованием различных методов масштабирования. Принимает данные и строковый аргумент, указывающий на метод масштабирования. Возвращает масштабированные данные в соответствии с выбранным методом.

2) Исследование работы классификатора, обученного на данных разного размера.

Код тестирования:

train\_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]

accuracies = []

for train\_size in train\_sizes:

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

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

predictions = predict(model, X\_test)

accuracy = metrics.accuracy\_score(y\_test, predictions)

accuracies.append(accuracy)

# Печать результатов

for i, train\_size in enumerate(train\_sizes):

print(f"Train Size: {train\_size}, Accuracy: {accuracies[i]}")

Оценка точности классификатора:

|  |  |
| --- | --- |
| train\_size | Точность |
| 0.1 | 0.3788819875776397 |
| 0.3 | 0.8 |
| 0.5 | 0.8426966292134831 |
| 0.7 | 0.8148148148148148 |
| 0.9 | 0.7222222222222222 |

Пояснение: train\_size = 0.1 – при таком маленьком размере обучающего набора точность модели низкая, так как модель обучается на сильно ограниченном объеме данных. train\_size = 0.3 – с увеличением размера обучающего набора точность значительно повышается, так как модель обучается на большем количестве данных. train\_size = 0.5 – при подходящем размере обучающего набора модель показывает хорошую точность. train\_size = 0.7 – точность в данном случае может снижаться из-за увеличения объема тестового набора. train\_size = 0.9 – при очень большом размере обучающего набора модель может переобучиться, что приводит к снижению точности на тестовом наборе. Таким образом, оптимальный размер обучающего набора выбирается исходя из баланса между недообучением и переобучением модели.

3) Исследование работы классификатора, обученного с различными значениями n\_neighbors.

Код тестирования:

n\_neighbors\_list = [3, 5, 9, 15, 25]

accuracies = []

X\_train, X\_test, y\_train, y\_test = load\_data(train\_size=0.7) # Может потребоваться вставить реальные данные

for n\_neighbors in n\_neighbors\_list:

model = neighbors.KNeighborsClassifier(n\_neighbors=n\_neighbors)

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

predictions = model.predict(X\_test)

accuracy = metrics.accuracy\_score(y\_test, predictions)

accuracies.append(accuracy)

# Печать результатов

for i, n\_neighbors in enumerate(n\_neighbors\_list):

print(f"n\_neighbors: {n\_neighbors}, Accuracy: {accuracies[i]}")

Оценка точности классификатора:

|  |  |
| --- | --- |
| n\_neighbors | Точность |
| 3 | 0.8148148148148148 |
| 5 | 0.7962962962962963 |
| 9 | 0.7777777777777778 |
| 15 | 0.8148148148148148 |
| 25 | 0.7962962962962963 |

Пояснение: n\_neighbors = 3 – при использовании 3 ближайших соседей классификатор достигает точности примерно 81.48%. Это может значить, что для небольшого числа соседей модель более чувствительна к местным особенностям данных. n\_neighbors = 5 – при увеличении числа соседей до 5 точность уменьшилась незначительно до примерно 79.63%. Возможно, в этом случае модель стала более устойчивой к выбросам или шуму в данных. n\_neighbors = 9 – при использовании 9 соседей точность снизилась до примерно 77.78%. Это может означать, что при увеличении количества соседей модель становится более обобщенной и склонной к недообучению. n\_neighbors = 15 – при 15 соседях точность вернулась к уровню примерно 81.48%. Это может указывать на то, что в данном случае модель снова начинает учитывать более мелкие детали данных, что повышает точность. n\_neighbors = 25 – при дальнейшем увеличении количества соседей до 25 точность снова снизилась до 79.63%. Это может говорить о потере значимости индивидуальных особенностей данных из-за увеличения числа соседей. Таким образом, выбор оптимального значения n\_neighbors для модели k-ближайших соседей важен для достижения наидучшей точности классификации.

4) Исследование работы классификатора с предобработанными данными.

Код тестирования:

scalers = {

'StandardScaler': preprocessing.StandardScaler(),

'MinMaxScaler': preprocessing.MinMaxScaler(),

'MaxAbsScaler': preprocessing.MaxAbsScaler()

}

accuracies = []

X\_train, X\_test, y\_train, y\_test = load\_data(train\_size=0.7) # Может потребоваться вставить реальные данные

for scaler\_name, scaler in scalers.items():

# Применение скейлера к данным

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Обучение классификатора

model = neighbors.KNeighborsClassifier(n\_neighbors=5) # Выбрано значение n\_neighbors=5 для примера

model.fit(X\_train\_scaled, y\_train)

# Предсказание и оценка точности

predictions = model.predict(X\_test\_scaled)

accuracy = metrics.accuracy\_score(y\_test, predictions)

accuracies.append(accuracy)

# Печать результатов

for i, (scaler\_name, \_) in enumerate(scalers.items()):

print(f"{scaler\_name}, Accuracy: {accuracies[i]}")

Оценка точности классификатора:

|  |  |
| --- | --- |
| Тип scaler | Точность |
| StandardScaler | 0.7592592592592593 |
| MinMaxScaler | 0.7592592592592593 |
| MaxAbsScaler | 0.7962962962962963 |

Пояснение: StandardScaler – точность составила примерно 75.93%. StandardScaler стандартизирует данные путем удаления среднего значения и масштабирования до единичной дисперсии. Это хороший выбор, когда признаки в данных имеют нормальное распределение. MinMaxScaler – точность составила примерно 75.93%. MinMaxScaler масштабирует данные в диапазоне [0,1]. Этот метод хорош для моделей, которые требуют входных данных в определенном диапазоне. MaxAbsScaler – точность составила примерно 79.63%. MaxAbsScaler масштабирует данные по максимальному абсолютному значению в каждом признаке. Этот метод хорошо работает с разреженными данными, когда признаки имеют различные шкалы. Таким образом, выбор метода масштабирования данных влияет на точность размера модели. В данном случае, использование MaxAbsScaler привело к лучшей точности по сравнению с другими методами.

Разработанный программный код см. в приложении А.

## 

## Выводы

В ходе лабораторной работы было изучено введение в анализ данных, создана программа на языке программирования Python, в которой используются функции и встроенный набор данных из библиотеки sklearn.

# Приложение А Исходный код программы

# импорт необходимых модулей из библиотеки scikit-learn

from sklearn import datasets, model\_selection, neighbors, metrics, preprocessing

# функция для загрузки данных о вине и разделения их на обучающий и тестовый наборы

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

# загрузка данных о вине из встроенного набора данных

wine = datasets.load\_wine()

X = wine.data

y = wine.target

# разделение данных на обучающий и тестовый наборы

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, random\_state = 42, train\_size = train\_size)

return X\_train[:, :2], X\_test[:, :2], y\_train, y\_test

# функция для обучения модели классификации методом k-ближайших соседей

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

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

# функция для предсказания меток классов на тестовых данных

def predict(clf, X\_test):

return clf.predict(X\_test)

# функция для оценки точности модели с помощью метрики accuracy

def estimate(res, y\_test):

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

return round(accuracy, 3)

# функция для масштабирования данных с использованием различных методов

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

if mode == 'standard':

scaler = preprocessing.StandardScaler() # стандартизация данных

elif mode == 'minmax':

scaler = preprocessing.MinMaxScaler() # масштабирование на отрезок [0, 1]

elif mode == 'maxabs':

scaler = preprocessing.MaxAbsScaler() # масштабирование по максимальному абсолютному значению

else:

return None

return scaler.fit\_transform(data)