Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки

«Анализ данных и машинное обучение»

**Классификация сухих бобов с учетом их размеров и формы.**

Разработчики проекта:

(ФИО)

Спиряков Максим Дмитриевич

Пермь, 2024

**Оглавление**

[ПАСПОРТ ПРОЕКТА 3](#_heading=h.30j0zll)

[СОДЕРЖАНИЕ ПРОЕКТА 4](#_heading=h.1fob9te)

[Анализ проблемы исследования 4](#_heading=h.3znysh7)

[Исходные данные 4](#_heading=h.2et92p0)

[Реализация проекта 6](#_heading=h.tyjcwt)

[Этап 1. Подготовка данных к анализу 6](#_heading=h.3dy6vkm)

[Этап 2. Предварительный анализ данных 8](#_heading=h.1t3h5sf)

[Этап 3. Корреляционный анализ данных 12](#_heading=h.4d34og8)

[Этап 4. Моделирование 14](#_heading=h.2s8eyo1)

[Заключение 21](#_heading=h.3rdcrjn)

[Список использованных источников и литературы 22](#_heading=h.26in1rg)

[Приложения 23](#_heading=h.lnxbz9)

**ПАСПОРТ ПРОЕКТА**

**Название проекта:** Классификация сухих бобов с учетом их размеров и формы.

**Сведения об авторах:**Спиряков Максим Дмитриевич.

**Цель:**выполнить анализ данных о бобах и построить наилучшую модель зависимости класса бобов от размеров и формы, позволяющую выполнять классификацию с высокой точностью.

**Задачи:**

1. Выполнить анализ проблемы, обосновать ее актуальность.
2. Осуществить загрузку данных и подготовку их к анализу количественными методами.
3. Выполнить предварительный анализ данных, в том числе выявление и обработку выбросов, выполнить нормализацию данных и корреляционный анализ.
4. Реализовать различные алгоритмы машинного обучения для классификации, такие как логистическая регрессия, метод ближайших соседей (KNN), деревья решений, наивный байесовский классификатор и метод опорных векторов (SVC), затем выбрать наилучшую модель.
5. Выполнить интерпретацию полученных результатов и сделать выводы о достижении цели.

**Краткое описание проекта:**

Проект направлен на разработку нейронной сети для классификации сухих бобов с учетом их размеров и формы. Основной целью является создание инструмента, который поможет идентифицировать сухие бобы.

**Конкретные ожидаемые результаты:** получение наилучших моделей, для определения вида бобов.

**СОДЕРЖАНИЕ ПРОЕКТА**

**Анализ проблемы исследования**

Идентификация сухих бобов может помочь в сельском хозяйстве. В зависимости от крупности семян бобы делят на крупные, средние и мелкие. Это позволяет выбирать куда их использовать. Например, мелкие бобы используют для корма скота.

Также стоит отметить, что при доработке этот проект поможет определять степень зрелости бобов. Сейчас мы имеем распределение видов бобов по их размерам и форме. Если рассматриваемые бобы, а точнее если их размер и форма не входят ни в одну из известных групп, то можно сделать вывод, что бобы являются недозрелыми.

Разработка нейронной сети для классификации сухих бобов может значительно облегчить работу селекционера и поможет ему правильнее и быстрее идентифицировать бобы.

**Исходные данные**

В этом исследовании использовались семь различных видов сухих бобов, с учётом таких характеристик, как форма, размер, тип и структура, в соответствии с ситуацией на рынке. Была разработана система компьютерного зрения для распознавания семи различных зарегистрированных сортов сухих бобов со схожими характеристиками, чтобы обеспечить единообразную классификацию семян. Для модели классификации были сделаны снимки 13 611 зёрен 7 различных зарегистрированных сортов сухих бобов с помощью камеры высокого разрешения. Изображения зёрен, полученные с помощью системы компьютерного зрения, были подвергнуты сегментации и извлечению признаков, в результате чего из зёрен было получено 16 признаков: 12 размеров и 4 формы.

* Площадь (A): площадь зоны боба и количество пикселей в её границах.
* Периметр (P): окружность боба определяется как длина его границы.
* Длина большой оси (L): расстояние между концами самой длинной линии, которую можно провести через боб.
* Длина малой оси (l): самая длинная линия, которую можно провести через боб, стоя перпендикулярно большой оси.
* Соотношение сторон (K): определяет соотношение между L и l.
* Эксцентриситет (Ec): эксцентриситет эллипса, имеющего те же моменты, что и область.
* Выпуклая область (C): количество пикселей в наименьшем выпуклом многоугольнике, который может содержать область семени боба.
* Эквивалентный диаметр (Ed): диаметр окружности, имеющей ту же площадь, что и область семени боба.
* Экстент (Ex): отношение количества пикселей в ограничивающей рамке к площади семени боба.
* Прочность (S): также известна как выпуклость. Отношение количества пикселей в выпуклой оболочке к количеству пикселей в бобах.
* Округлость (R): рассчитывается по следующей формуле: (4piA)/(P^2)
* Компактность (CO): измеряет округлость объекта: Ed/L
* Коэффициент формы 1 (SF1)
* Коэффициент формы 2 (SF2)
* Коэффициент формы 3 (SF3)
* Коэффициент формы 4 (SF4)
* Класс (Секер, Барбунья, Бомбей, Кали, Дермозан, Хороз и Сира)

Необходимо проанализировать данные по бобам и определить, возможно ли описать зависимость классов от имеющихся характеристик различными методами.

**Реализация проекта**

**Этап 1. Подготовка данных к анализу**

Загрузим данные в датафрейм и подключим необходимые библиотеки:

import numpy as np

import pandas as pd

import seaborn as sns

sns.set\_palette('husl')

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

import mpl\_toolkits.mplot3d

from sklearn import datasets

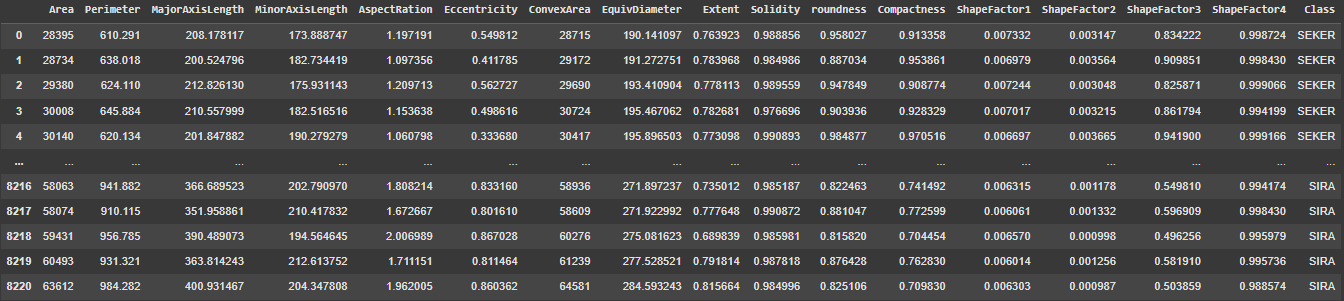
from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

Загрузим датасет:

dataset = pd.read\_csv('/content/Dry\_Bean.csv', delimiter=',')

dataset

****

**Рисунок 1.** Исходный датафрейм.

Проверим есть ли пропущенные значения:

dataset.isnull().sum()

Пропущенных значений нет, посмотрим количество значений каждого класса:

dataset['Class'].value\_counts()



**Рисунок 2.** Количество значений каждого класса.

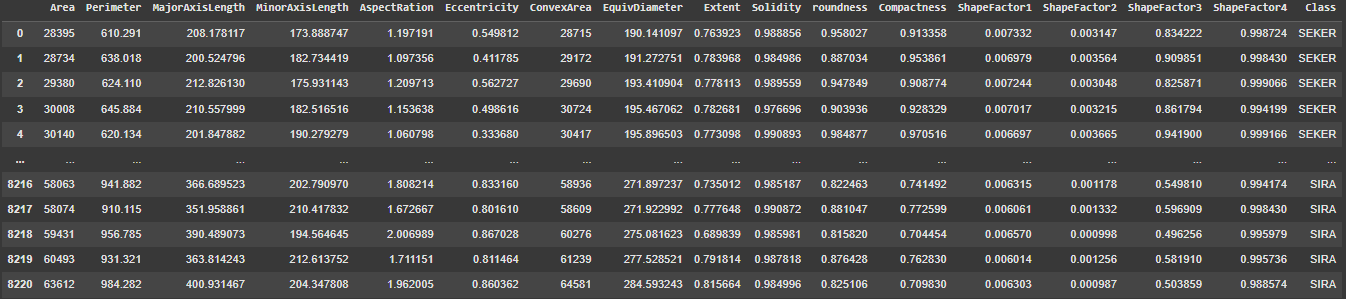
Ограничимся 4 классами для более успешной классификации. Выбранные классы: SIRA, SEKER, HOROZ, CALI.

dataset = dataset.loc[dataset['Class'] != 'DERMASON'].reset\_index(drop=True)

dataset = dataset.loc[dataset['Class'] != 'BARBUNYA'].reset\_index(drop=True)

dataset = dataset.loc[dataset['Class'] != 'BOMBAY'].reset\_index(drop=True)

dataset



**Рисунок 3.** Датасет после удаления классов.

Нормализуем наш датасет, используя MinMaxScaler.

scaler = MinMaxScaler()

scaled = scaler.fit\_transform(dataset.drop(columns=['Class']))

scaled\_df = pd.DataFrame(scaled, columns=dataset.drop(columns=['Class']).columns)

class\_column = dataset["Class"]

scaled\_dataset = pd.concat([scaled\_df,class\_column], axis = 1)

scaled\_dataset



**Рисунок 4.** Датасет после нормализации.

**Этап 2. Предварительный анализ данных**

Построим скрипичные графики.

sns.violinplot(y='Class', x='Area', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='Perimeter', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='MajorAxisLength', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='MinorAxisLength', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='AspectRation', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='Eccentricity', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='ConvexArea', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='EquivDiameter', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='Extent', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='Solidity', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='roundness', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='Compactness', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='ShapeFactor1', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='ShapeFactor2', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='ShapeFactor3', data=scaled\_dataset, inner='quartile')

plt.show()

sns.violinplot(y='Class', x='ShapeFactor4', data=scaled\_dataset, inner='quartile')

plt.show()

|  |  |
| --- | --- |
|  |  |
|  |  |

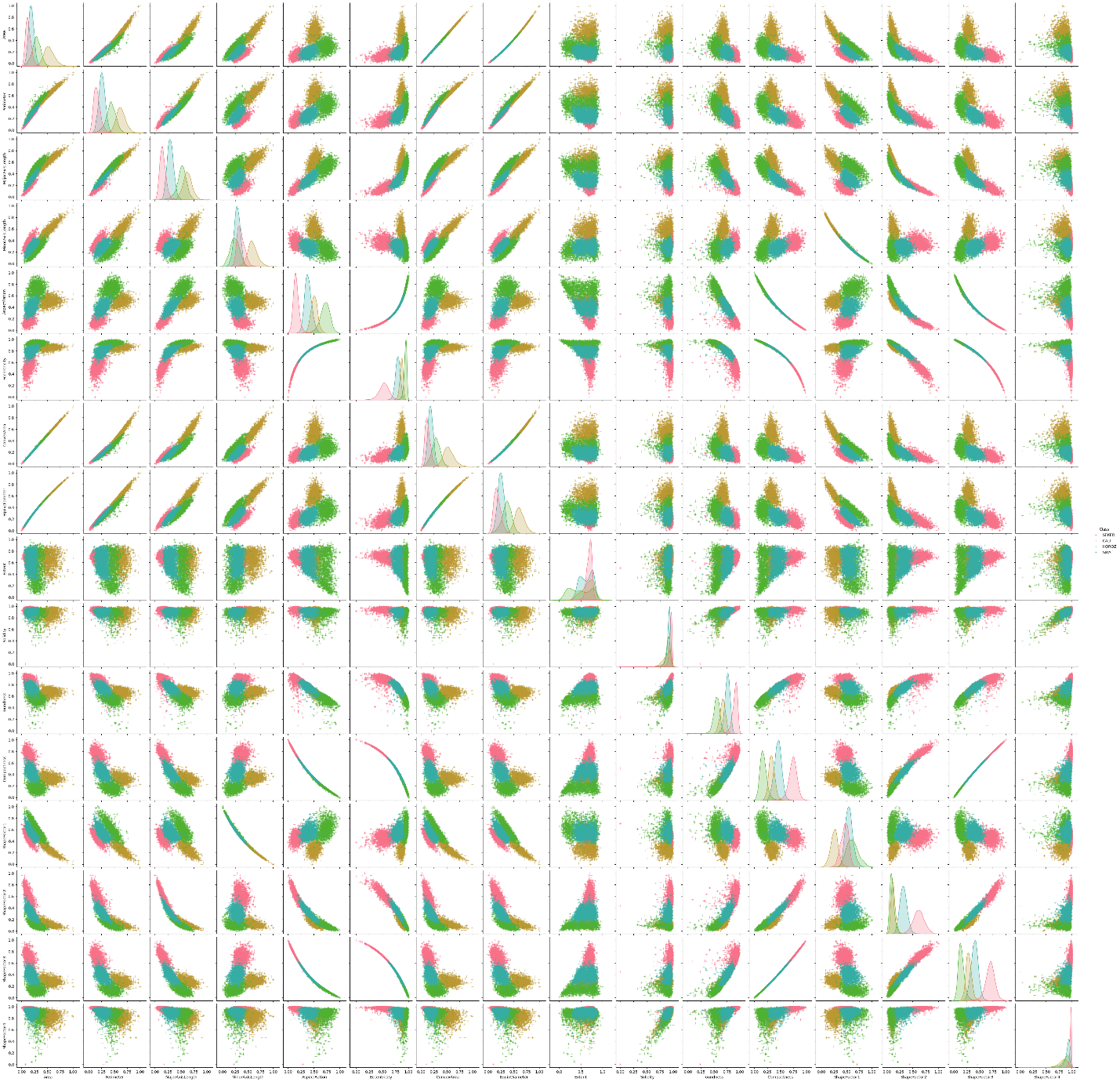
**Рисунок 5.** Некоторые скрипичные графики.

Все характеристики, представленные на рисунке 5, в отличие от остальных, имеют больший разброс значений для четырех классов бобов.

Построим парные графики.

sns.pairplot(scaled\_dataset, hue='Class', markers='+')

plt.show()



**Рисунок 6.** Парные графики (более наглядное представление находится в коде).

**Этап 3. Корреляционный анализ данных**

***Корреляционный анализ*** – это совокупность методов оценивания степени тесноты статистической связи между анализируемыми переменными.

Выполним корреляционный анализ данных с помощью матрицы корреляции. В случае нормальности всех данных следует использовать коэффициент корреляции Пирсона, в противном случае – ранговые коэффициенты корреляции Спирмена.

**Парный коэффициент** корреляции характеризует взаимосвязь двух переменных на фоне действия остальных показателей и является самым распространенным показателем тесноты связи при статистическом анализе данных.

Парный коэффициент корреляции между количественными случайными переменными и носит название *выборочного коэффициента корреляции* Пирсона (*sample correlation coefficient*) (или просто коэффициента корреляции) и находится по формуле

где и  — *выборочные дисперсии (sample variances*) переменных  и , а — *выборочная ковариация* или выборочный ковариационный момент, и соответствующие *средние (means)* определяются по формулам



Коэффициент корреляции обладает следующими свойствами:

1. Принимает значения от –1 до +1.
2. Если , то связь между переменными  и  считается сильной. Если , то связь слабая.
3. Если , то корреляционное поле наблюдений представляет собой совокупность точек, которые можно расположить на одной прямой. Знак «+» свидетельствует о прямой линейной зависимости между переменными и , а знак «» − об обратной линейной зависимости.

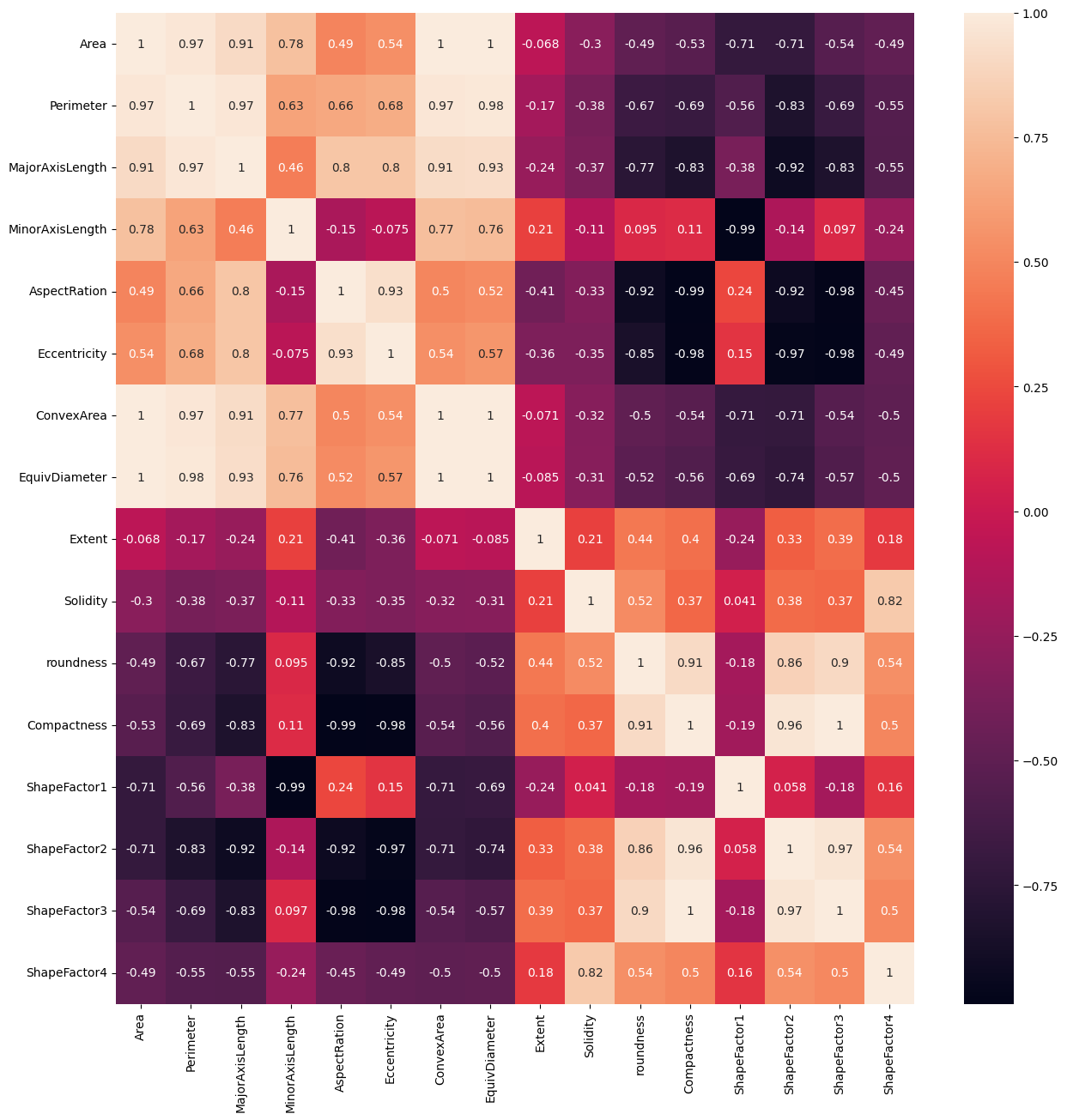
При  линейная корреляционная связь отсутствует.

Построим тепловую карту матрицу корреляции.

plt.figure(figsize=(15,15))

sns.heatmap(scaled\_dataset.corr(numeric\_only = True), annot=True)

plt.show()



**Рисунок 7.** Тепловая карта матрицы корреляции.

Так как все характеристики являются в основном размерами (в какой−то степени), следовательно нам необходимо исключить признаки, у которых сильная зависимость между собой. Проанализировав матрицу корреляций, выбор пал на такие характеристики, как: «MajorAxisLength», «Extent», «ShapeFactor1».

**Этап 4. Моделирование**

Определим признаки и целевую переменную.

X = scaled\_dataset.drop(columns=["Area",  "Perimeter",  "MinorAxisLength",  "AspectRation", "Eccentricity", "ConvexArea", "EquivDiameter", "Solidity",  "roundness",  "Compactness",  'ShapeFactor2', "ShapeFactor3", "ShapeFactor4", "Class"])

y = scaled\_dataset['Class']

print(f'X shape: {X.shape} | y shape: {y.shape} ')

Разобьём данные на обучающую и тестовую выборки, используя соотношение 80 на 20:

y\_mapped = y.map({'SIRA': 0, 'SEKER': 1, 'HOROZ': 2, 'CALI': 3})

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_mapped, test\_size=0.20, random\_state=15, stratify=y\_mapped)

stats.ttest\_ind (a=y\_train, b=y\_test)

Проведем оценку следующих моделей и выберем лучшие из них:

1. **Логистическая регрессия (LR)**:
   * LogisticRegression — это линейная модель, которая используется для предсказания вероятности принадлежности объекта к определенному классу. Она хорошо подходит для бинарной классификации, но также может быть адаптирована для многоклассовых задач.
2. **Линейный дискриминантный анализ (LDA)**:
   * LinearDiscriminantAnalysis() — это статистический метод, который ищет линейные комбинации признаков, которые лучше всего разделяют классы.
3. **Метод ближайших соседей (KNN)**:
   * KNeighborsClassifier() — это не параметрическая модель, которая классифицирует объекты на основе классов их ближайших соседей в пространстве признаков.
4. **Дерево решений (CART)**:
   * DecisionTreeClassifier() — это модель, которая принимает решения, основываясь на последовательных вопросах о значениях признаков.
5. **Наивный байесовский классификатор (NB)**:
   * GaussianNB() — это вероятностная модель, которая основана на применении теоремы Байеса с предположением о независимости признаков.
6. **Метод опорных векторов (SVC)**:
   * SVC() — это модель, которая находит гиперплоскость, разделяющую классы в многомерном пространстве.

Оценка проводится методом кросс-валидации:

# Модели

models = []

models.append(('LR', LogisticRegression(solver='lbfgs', max\_iter=1000)))

models.append(('LDA', LinearDiscriminantAnalysis()))

models.append(('KNN', KNeighborsClassifier()))

models.append(('CART', DecisionTreeClassifier()))

models.append(('NB', GaussianNB()))

models.append(('SVC', SVC(gamma='auto')))

# Оценка метрик

results = []

model\_names = []

for name, model in models:

  kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

  cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='accuracy')

  results.append(cv\_results)

  model\_names.append(name)

  print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))

В итоге получили следующее:

LR: 0.957578 (0.009825)

LDA: 0.954993 (0.010162)

KNN: 0.955144 (0.008879)

CART: 0.935376 (0.012427)

NB: 0.938567 (0.009009)

SVC: 0.958642 (0.009449)

Среди всех моделей наилучший результат у LR, SVC, KNN. Приступим к обучению и тестированию.

Рассмотрим SVC.

sk\_SVC = SVC()

sk\_SVC.fit(X\_train, y\_train)

sk\_SVC\_pred\_res = sk\_SVC.predict(X\_test)

sk\_SVC\_accuracy = accuracy\_score(y\_test, sk\_SVC\_pred\_res)

print(f'sk SVC accuracy: {sk\_SVC\_accuracy}')

print(f'sk SVC  prediction: {sk\_SVC\_pred\_res}')

from mlxtend.plotting import plot\_decision\_regions

y\_s = y\_train

X\_2d= X\_train[['MajorAxisLength', 'ShapeFactor1']].values

y\_2d = y\_s.values

sk\_SVC1 = SVC()

sk\_SVC1.fit(X\_2d, y\_2d)

plt.title('SVC surface with original features')

plot\_decision\_regions(

      X=X\_2d,

      y=y\_2d,

      clf=sk\_SVC1)

Посмотрим метрики на тестовом датасете.

from sklearn import model\_selection, preprocessing, feature\_selection, ensemble, linear\_model, metrics, decomposition

accuracy = metrics.accuracy\_score(y\_test, sk\_LR\_pred\_res)

recall = metrics.recall\_score(y\_test, sk\_LR\_pred\_res, average="weighted")

precision = metrics.precision\_score(y\_test, sk\_LR\_pred\_res, average="weighted")

print("Recall (all 1s predicted right):", round(recall,2))

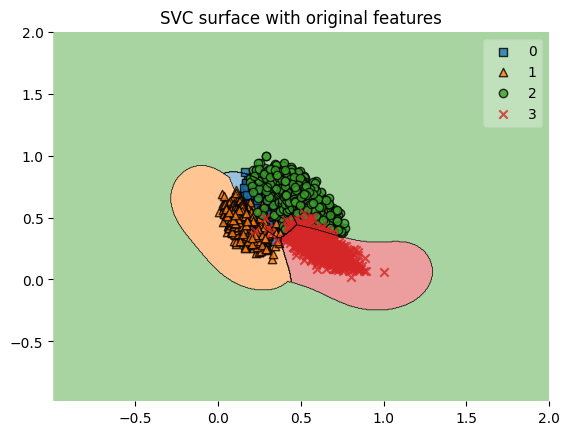
print("Precision (confidence when predicting a 1):", round(precision,2))

print("Detail:")

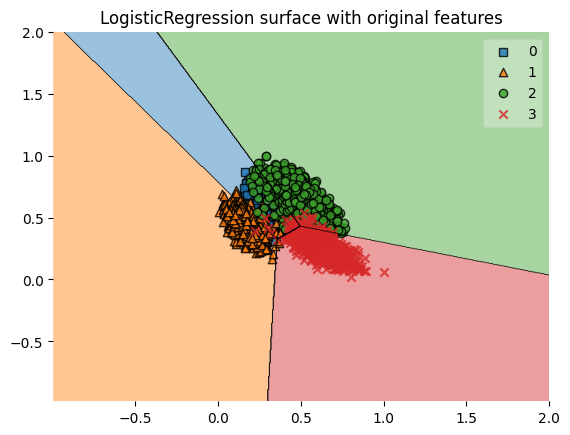
print(metrics.classification\_report(y\_test, sk\_LR\_pred\_res, target\_names=[str(i) for i in np.unique(y\_test)]))

Для других моделей код аналогичный.

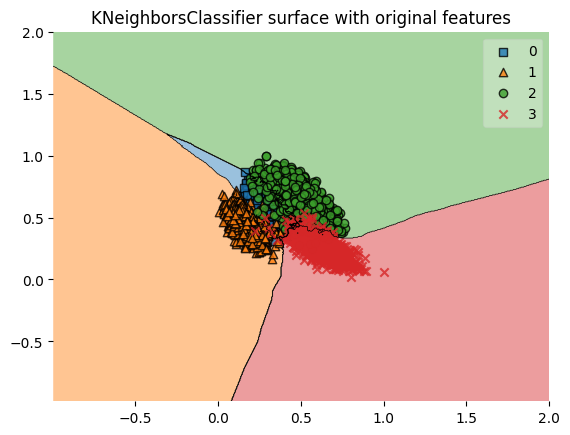
Результаты:



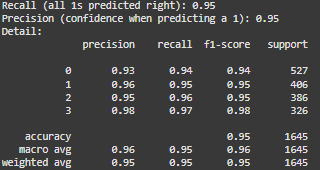
**Рисунок 8.** Классификация SVC.



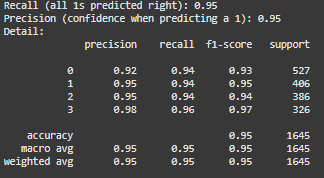
**Рисунок 9.** Классификация LR.



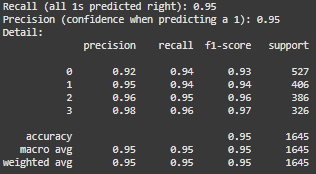
**Рисунок 10.** Классификация KNN.



**Рисунок 11.** Метрики на тестовом датасете SVC.



**Рисунок 12.** Метрики на тестовом датасете LR.



**Рисунок 13.** Метрики на тестовом датасете KNN.

Все модели показали высокие результаты ≈ 95% точности, что говорит о хорошем разделении на классы. Но стоит учитывать, что все эти модели могут иногда ошибаться при классификации.

**Заключение**

В процессе написания проекта были успешно выполнены все четыре этапа. На первом этапе был загружен и нормализован датасет, который включает в себя характеристики сухих бобов. На втором этапе были проанализированы данные, построены различные графики. На третьем этапе был проведен корреляционный анализ, по итогу которого были выбраны три признака, по которым проводилась классификация. На четвертом этапе проведена оценка шести моделей, по результатам которой лучшими оказались SCV, LR и KNN.

Данный проект реализован с помощью методов машинного для классификации сухих бобов при учете их размеров и формы. Результаты правильной, высокоточной классификации указывают на то, что выбранные признаки хорошо совместимы для разделения бобов по классам.

**Список использованных источников и литературы**

1. Сузи Р.А. Язык программирования Python: учебное пособие / Р.А. Сузи. – 3-е изд. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. – 350 c. // Электронно-библиотечная система IPR BOOKS: [сайт].
2. Волкова В.М., Программные системы статистического анализа. Обнаружение закономерностей в данных с использованием системы R и языка Python [Электронный ресурс]: учебное пособие / Волкова В.М. - Новосибирск : Изд-во НГТУ, 2017. - 74 с. - ISBN 978-5-7782-3183-2 - Режим доступа: http://www.studentlibrary.ru/book/ISBN9785778231832.html
3. Капитонов, А. А. (2017). Обработка и анализ данных с использованием Python и R. М.: БХВ-Петербург.
4. Федоров Д.Ю. Программирование на языке высокого уровня Python: учебное пособие / Д.Ю. Федоров. – 2-е изд.– М.: Юрайт, 2020. – 161 с.
5. Маккини У. Python и анализ данных / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 540 с.

**Приложения**

Приложение 1

Программный код

[301]

*# Импортируем библиотеки*  
  
**import** numpy **as** np  
**import** pandas **as** pd  
**import** seaborn **as** sns  
sns.set\_palette('husl')  
**import** matplotlib.pyplot **as** plt  
%matplotlib inline  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.model\_selection **import** cross\_val\_score  
**from** sklearn.model\_selection **import** StratifiedKFold  
**from** sklearn.metrics **import** classification\_report  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn.linear\_model **import** LogisticRegression  
**from** sklearn.tree **import** DecisionTreeClassifier  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis  
**from** sklearn.naive\_bayes **import** GaussianNB  
**from** sklearn.svm **import** SVC  
**import** mpl\_toolkits.mplot3d  
**from** sklearn **import** datasets  
**from** sklearn.cluster **import** KMeans  
**from** sklearn.preprocessing **import** MinMaxScaler

[302]

*# Работа с датасетом*  
dataset = pd.read\_csv('/content/Dry\_Bean.csv', delimiter=',')

[303]

dataset.shape

(13611, 17)

[304]

dataset.head(10)

[305]

dataset.info()

[306]

dataset.isnull().sum()

[307]

dataset.describe()

[308]

dataset['Class'].unique()

array(['SEKER', 'BARBUNYA', 'BOMBAY', 'CALI', 'HOROZ', 'SIRA', 'DERMASON'],  
 dtype=object)

[309]

dataset['Class'].value\_counts()

[312]

*# Удаление строк с бобами: DERMASON, BARBUNYA, BOMBAY*  
  
dataset = dataset.loc[dataset['Class'] != 'DERMASON'].reset\_index(drop=True)  
dataset = dataset.loc[dataset['Class'] != 'BARBUNYA'].reset\_index(drop=True)  
dataset = dataset.loc[dataset['Class'] != 'BOMBAY'].reset\_index(drop=True)

[313]

dataset

[315]

*# Нормализация*  
scaler = MinMaxScaler()  
  
scaled = scaler.fit\_transform(dataset.drop(columns=['Class']))  
  
scaled\_df = pd.DataFrame(scaled, columns=dataset.drop(columns=['Class']).columns)  
  
class\_column = dataset["Class"]  
scaled\_dataset = pd.concat([scaled\_df,class\_column], axis = 1)  
scaled\_dataset

[316]

*# Матрица корреляций*  
plt.figure(figsize=(15,15))  
sns.heatmap(scaled\_dataset.corr(numeric\_only = True), annot=True)  
plt.show()

[317]

*# Скрипичные графики*  
  
sns.violinplot(y='Class', x='Area', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='Perimeter', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='MajorAxisLength', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='MinorAxisLength', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='AspectRation', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='Eccentricity', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='ConvexArea', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='EquivDiameter', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='Extent', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='Solidity', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='roundness', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='Compactness', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='ShapeFactor1', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='ShapeFactor2', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='ShapeFactor3', data=scaled\_dataset, inner='quartile')  
plt.show()  
sns.violinplot(y='Class', x='ShapeFactor4', data=scaled\_dataset, inner='quartile')  
plt.show()

[318]

*# Парные графики*  
  
sns.pairplot(scaled\_dataset, hue='Class', markers='+')  
plt.show()

[319]

*# Разделение набора данных. В X: 'MajorAxisLength', 'Extent', 'ShapeFactor1'*  
  
X = scaled\_dataset.drop(columns=["Area",    "Perimeter",    "MinorAxisLength",  "AspectRation", "Eccentricity", "ConvexArea",   "EquivDiameter", "Solidity",    "roundness",    "Compactness",  'ShapeFactor2', "ShapeFactor3", "ShapeFactor4", "Class"])  
y = scaled\_dataset['Class']  
print(**f**'X shape: {X.shape} | y shape: {y.shape} ')

X shape: (8221, 3) | y shape: (8221,)

[329]

**import** scipy.stats **as** stats  
  
y\_mapped = y.map({'SIRA': 0, 'SEKER': 1, 'HOROZ': 2, 'CALI': 3})  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_mapped, test\_size=0.30, shuffle=False)  
  
stats.ttest\_ind (a=y\_train, b=y\_test)

TtestResult(statistic=108.32930779214476, pvalue=0.0, df=8219.0)

[330]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_mapped, test\_size=0.20, random\_state=15, stratify=y\_mapped)  
  
stats.ttest\_ind (a=y\_train, b=y\_test)

TtestResult(statistic=-0.008663348971295661, pvalue=0.9930879443361207, df=8219.0)

[340]

*# Модели*  
models = []  
models.append(('LR', LogisticRegression(solver='lbfgs', max\_iter=1000)))  
models.append(('LDA', LinearDiscriminantAnalysis()))  
models.append(('KNN', KNeighborsClassifier()))  
models.append(('CART', DecisionTreeClassifier()))  
models.append(('NB', GaussianNB()))  
models.append(('SVC', SVC(gamma='auto')))  
  
*# Оценка метрик*  
results = []  
model\_names = []  
**for** name, model **in** models:  
  kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)  
  cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='accuracy')  
  results.append(cv\_results)  
  model\_names.append(name)  
  print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))

[341]

*# Рассмотрим SVC*  
  
sk\_SVC = SVC()  
sk\_SVC.fit(X\_train, y\_train)  
sk\_SVC\_pred\_res = sk\_SVC.predict(X\_test)  
sk\_SVC\_accuracy = accuracy\_score(y\_test, sk\_SVC\_pred\_res)  
  
print(**f**'sk SVC accuracy: {sk\_SVC\_accuracy}')  
print(**f**'sk SVC  prediction: {sk\_SVC\_pred\_res}')

sk SVC accuracy: 0.9531914893617022  
sk SVC prediction: [1 1 3 ... 2 3 0]

[342]

*# Классификация SVC*  
  
**from** mlxtend.plotting **import** plot\_decision\_regions  
  
y\_s = y\_train  
X\_2d= X\_train[['MajorAxisLength',   'ShapeFactor1']].values  
  
y\_2d = y\_s.values  
  
sk\_SVC1 = SVC()  
sk\_SVC1.fit(X\_2d, y\_2d)  
  
plt.title('SVC surface with original features')  
plot\_decision\_regions(  
        X=X\_2d,  
        y=y\_2d,  
        clf=sk\_SVC1)

[343]

*# Рассмотрим метрики на тестовом датасете*  
  
**from** sklearn **import** model\_selection, preprocessing, feature\_selection, ensemble, linear\_model, metrics, decomposition  
  
accuracy = metrics.accuracy\_score(y\_test, sk\_SVC\_pred\_res)  
  
recall = metrics.recall\_score(y\_test, sk\_SVC\_pred\_res, average="weighted")  
precision = metrics.precision\_score(y\_test, sk\_SVC\_pred\_res, average="weighted")  
print("Recall (all 1s predicted right):", round(recall,2))  
print("Precision (confidence when predicting a 1):", round(precision,2))  
print("Detail:")  
print(metrics.classification\_report(y\_test, sk\_SVC\_pred\_res, target\_names=[str(i) **for** i **in** np.unique(y\_test)]))

[344]

*# Рассмотрим LogisticRegression*  
  
sk\_LR = LogisticRegression()  
sk\_LR.fit(X\_train, y\_train)  
sk\_LR\_pred\_res = sk\_LR.predict(X\_test)  
sk\_LR\_accuracy = accuracy\_score(y\_test, sk\_LR\_pred\_res)  
  
print(**f**'sk LR accuracy: {sk\_LR\_accuracy}')  
print(**f**'sk LR  prediction: {sk\_LR\_pred\_res}')

sk LR accuracy: 0.9458966565349544  
sk LR prediction: [1 1 3 ... 2 3 0]

[345]

*# Классификация LogisticRegression*  
  
**from** mlxtend.plotting **import** plot\_decision\_regions  
  
y\_s = y\_train  
X\_2d= X\_train[['MajorAxisLength',   'ShapeFactor1']].values  
  
y\_2d = y\_s.values  
  
sk\_LR1 = LogisticRegression()  
sk\_LR1.fit(X\_2d, y\_2d)  
  
plt.title('LogisticRegression surface with original features')  
plot\_decision\_regions(  
        X=X\_2d,  
        y=y\_2d,  
        clf=sk\_LR1)

[346]

*# Рассмотрим метрики на тестовом датасете*  
  
**from** sklearn **import** model\_selection, preprocessing, feature\_selection, ensemble, linear\_model, metrics, decomposition  
  
accuracy = metrics.accuracy\_score(y\_test, sk\_LR\_pred\_res)  
  
recall = metrics.recall\_score(y\_test, sk\_LR\_pred\_res, average="weighted")  
precision = metrics.precision\_score(y\_test, sk\_LR\_pred\_res, average="weighted")  
print("Recall (all 1s predicted right):", round(recall,2))  
print("Precision (confidence when predicting a 1):", round(precision,2))  
print("Detail:")  
print(metrics.classification\_report(y\_test, sk\_LR\_pred\_res, target\_names=[str(i) **for** i **in** np.unique(y\_test)]))

[347]

*# Рассмотрим KNeighborsClassifier*  
  
sk\_KNN = KNeighborsClassifier()  
sk\_KNN.fit(X\_train, y\_train)  
sk\_KNN\_pred\_res = sk\_KNN.predict(X\_test)  
sk\_KNN\_accuracy = accuracy\_score(y\_test, sk\_KNN\_pred\_res)  
  
print(**f**'sk KNN accuracy: {sk\_KNN\_accuracy}')  
print(**f**'sk KNN  prediction: {sk\_KNN\_pred\_res}')

sk KNN accuracy: 0.948936170212766  
sk KNN prediction: [1 1 3 ... 2 3 0]

[348]

*# Классификация KNeighborsClassifier*  
  
**from** mlxtend.plotting **import** plot\_decision\_regions  
  
y\_s = y\_train  
X\_2d= X\_train[['MajorAxisLength',   'ShapeFactor1']].values  
  
y\_2d = y\_s.values  
  
sk\_KNN1 = KNeighborsClassifier()  
sk\_KNN1.fit(X\_2d, y\_2d)  
  
plt.title('KNeighborsClassifier surface with original features')  
plot\_decision\_regions(  
        X=X\_2d,  
        y=y\_2d,  
        clf=sk\_KNN1)

[349]

*# Рассмотрим метрики на тестовом датасете*  
  
**from** sklearn **import** model\_selection, preprocessing, feature\_selection, ensemble, linear\_model, metrics, decomposition  
  
accuracy = metrics.accuracy\_score(y\_test, sk\_KNN\_pred\_res)  
  
recall = metrics.recall\_score(y\_test, sk\_KNN\_pred\_res, average="weighted")  
precision = metrics.precision\_score(y\_test, sk\_KNN\_pred\_res, average="weighted")  
print("Recall (all 1s predicted right):", round(recall,2))  
print("Precision (confidence when predicting a 1):", round(precision,2))  
print("Detail:")  
print(metrics.classification\_report(y\_test, sk\_KNN\_pred\_res, target\_names=[str(i) **for** i **in** np.unique(y\_test)]))