**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Основы машинного обучения»**

**Тема: Классификация**

**Вариант 1**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Изучить различные методы классификации данных.

**Задание.**

1. **Загрузка данных**
   1. Загрузите данные вашего варианта. Учтите, что метки классов являются текстом.
   2. Визуализируйте данные при помощи диаграммы рассеяния с выделением различных классов ней.
   3. Оцените сбалансированность классов
   4. Проведите предобработку данных при необходимости
   5. Разделите выборку на обучающую и тестовую. На обучающей проводите обучение модели, на тестовой расчет значений метрик.
2. **kNN**
   1. Проведите классификацию методом k ближайших соседей, подобрав параметры: количество соседей, необходимость взвешивани. Постройте графики зависимости точности (Accuracy) в зависимости от количества соседей (по 1-й линии для взвешенного и не взвешенного метода)
   2. Постройте изображение с границами принятия решения ответив на них точки классов разным цветом. Сделайте выводы о качестве классификации на основе полученного изображения.
   3. Постройте таблицу ошибок для полученных результатов. Сделайте выводы о классификации на основе таблицы ошибок.
   4. Рассчитайте значения Precision,Recall,F1 для полученных результатов. Сопоставьте с таблицей ошибок.
   5. Постройте изображение ROC-кривой и рассчитайте значение AUC для полученных результатов. Сделайте выводы о классификации по полученному изображение и значению AUC.
3. **Логистическая регрессия**
   1. Проведите классификацию методом логистической регрессии при различных параметрах: без регуляризации, с l1 регуляризацией, c l2 регуляризацией. Постройте столбчатую диаграмму зависимости точности (Accuracy) от наличия регуляризации. Дальнейшие пункты 3.\* выполняйте для лучшего параметра.
   2. Постройте изображение с границами принятия решения ответив на них точки классов разным цветом. Сделайте выводы о качестве классификации на основе полученного изображения.
   3. Постройте таблицу ошибок для полученных результатов. Сделайте выводы о классификации на основе таблицы ошибок.
   4. Рассчитайте значения Precision,Recall,F1 для полученных результатов. Сопоставьте с таблицей ошибок.
   5. Постройте изображение ROC-кривой и рассчитайте значение AUC для полученных результатов. Сделайте выводы о классификации по полученному изображение и значению AUC.
4. **Метод опорных векторов**
   1. Проведите классификацию методом опорных векторах при различных параметрах ядра: “linear”, “poly” (нужно выбрать степень), “rbf”. Постройте столбчатую диаграмму зависимости точности (Accuracy) от вида ядра. Дальнейшие пункты 4.\* выполняйте для лучшего параметра.
   2. Постройте изображение с границами принятия решения ответив на них точки классов разным цветом. Сделайте выводы о качестве классификации на основе полученного изображения.
   3. Постройте таблицу ошибок для полученных результатов. Сделайте выводы о классификации на основе таблицы ошибок.
   4. Рассчитайте значения Precision,Recall,F1 для полученных результатов. Сопоставьте с таблицей ошибок.
   5. Постройте изображение ROC-кривой и рассчитайте значение AUC для полученных результатов. Сделайте выводы о классификации по полученному изображение и значению AUC.
5. **Решающие деревья**
   1. Проведите классификацию используя решающие деревья, подобрав параметры при которых получается лучшее обобщение (максимальная глубина/максимальное количество листьев/метрика загрязнения и т.д.).
   2. Постройте изображение с границами принятия решения ответив на них точки классов разным цветом. Сделайте выводы о качестве классификации на основе полученного изображения.
   3. Постройте таблицу ошибок для полученных результатов. Сделайте выводы о классификации на основе таблицы ошибок.
   4. Рассчитайте значения Precision,Recall,F1 для полученных результатов. Сопоставьте с таблицей ошибок.
   5. Постройте изображение ROC-кривой и рассчитайте значение AUC для полученных результатов. Сделайте выводы о классификации по полученному изображение и значению AUC.
   6. Визуализируйте полученное дерево решений. Сделайте выводы о правилах в узлах дерева. Сопоставьте их с полученными границами принятия решений.
6. **Выбор классификатора**
   1. Постройте таблицу с метриками Precision, Recall, AUC для полученных результатов каждым классификатором.
   2. Сделайте выводы о том, какие классификаторы лучше всего подходят для вашего набора данных и в каких случаях.

**Выполнение работы.**

1. Загрузка данных
   1. Загрузим данные (табл. 1.1)

Таблица 1.1 – первые 5 строк файла lab4\_6

|  | **X1** | **X2** | **Class** |
| --- | --- | --- | --- |
| **0** | 0.689551 | 0.900750 | MO |
| **1** | 0.283680 | 0.713806 | MO |
| **2** | 0.122529 | -0.646096 | MO |
| **3** | 0.603928 | -0.485924 | MO |
| **4** | -0.298863 | 0.390297 | ON |

* 1. Визуализируем данные при помощи диаграммы рассеяния с выделением различных классов ней (рис. 1.2).

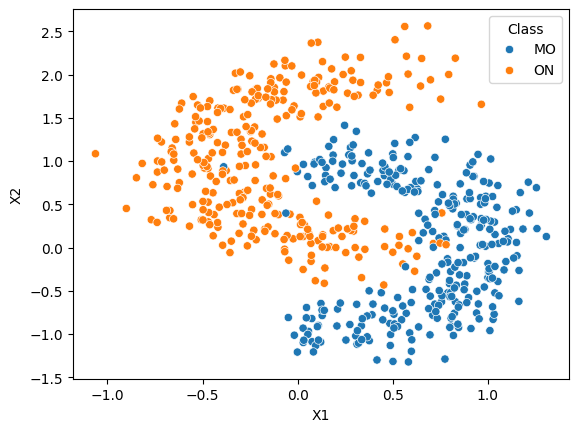


Рисунок 1.2 – диаграмма рассеяния lab4\_6

Из рисунка 1.2 видно, что имеется 2 класса, данные имеют нелинейную форму.

* 1. Оценим сбалансированность классов.

Из рисунка 1.2 можно увидеть, что классы примерно равны по количеству наблюдений. В дополнение посчитаем точные доли классов (листинг 1.3.1).

Листинг 1.3.1 – подсчет долей классов.

| class\_counts = df['Class'].value\_counts()  total\_samples = df.shape[0]  class\_0\_ratio = class\_counts[0] / total\_samples  class\_1\_ratio = class\_counts[1] / total\_samples  print("MO:", class\_0\_ratio)  print("ON:", class\_1\_ratio) |
| --- |

Доля МО составляет 0.5008319467554077, доля ON – 0.49916805324459235. Это подтверждает, что классы относительно сбалансированы, количество наблюдений для MO чуть-чуть больше, чем в ON.

* 1. Проведем предобработку данных.

Для этого заменим текстовые названия классов на числовые значения и удалим повторяющиеся и пустые записи (таблица 1.4.1, листинг 1.4.1). Также проведем нормализацию данных с помощью StandartScaler (листинг 1.4.2).

Таблица 1.4.1 – первые 5 строк файла lab4\_6 после предобработки

|  | **X1** | **X2** | **Class** |
| --- | --- | --- | --- |
| **0** | 0.689551 | 0.900750 | 0 |
| **1** | 0.283680 | 0.713806 | 0 |
| **2** | 0.122529 | -0.646096 | 0 |
| **3** | 0.603928 | -0.485924 | 0 |
| **4** | -0.298863 | 0.390297 | 1 |

Листинг 1.4.1 – предобработка данных.

| label\_encoders = {}  for column in df.columns:  if df[column].dtype == 'object':  label\_encoders[column] = LabelEncoder()  df[column] = label\_encoders[column].fit\_transform(df[column])  df.dropna(inplace=True)  df.drop\_duplicates(inplace=True) |
| --- |

Листинг 1.4.2 – нормировка данных.

| X = df[["X1", "X2"]].to\_numpy()  y = df["Class"].to\_numpy()  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X) |
| --- |

* 1. Разделим выборку на обучающую и тестовую(листинг 1.5, рис. 1.5).

Листинг 1.5 – разбиение выборки на обучающую и тестовую.

| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3, shuffle = False) |
| --- |

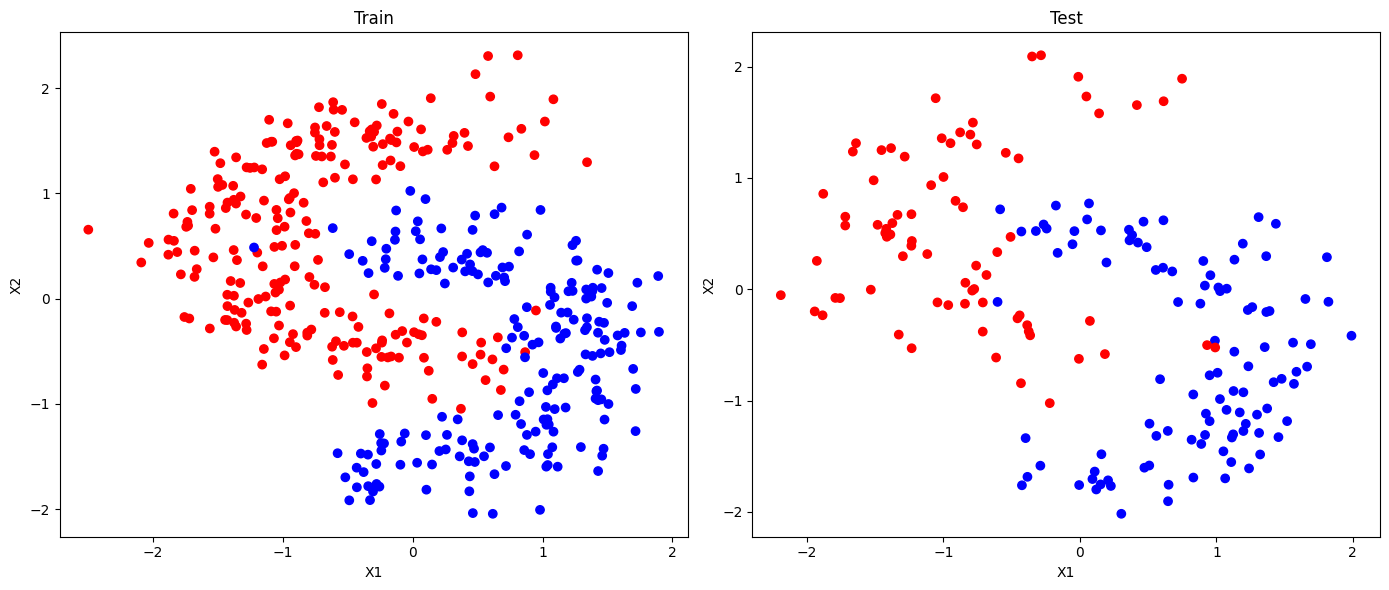


Рисунок 1.5 – диаграммы рассеяния обучающей и тестовой выборок.

Из рисунка 1.5 можно увидеть, что обучающая и тестовая выборки совпадают. Данные находятся примерно в интервале от -2 до 2.

1. kNN
   1. Проведем классификацию методом k ближайших соседей, подобрав параметры: количество соседей, необходимость взвешивания. Построим графики зависимости точности (Accuracy) в зависимости от количества соседей (по 1-й линии для взвешенного и не взвешенного метода)

Подберем параметры и построим графики зависимости точности от количества соседей для взвешенного и невзвешенного методов (листинг 2.1, рис. 2.1).

Листинг 2.1 – тестирование разного количества соседей для взвешенного и невзвешенного методов.

| accuracy\_unweighted = []  accuracy\_weighted = []  neighbors\_range = range(1, 11)  for k in neighbors\_range:  knn\_unweighted = KNeighborsClassifier(n\_neighbors=k)  knn\_unweighted.fit(X\_train, y\_train) accuracy\_unweighted.append(knn\_unweighted.score(X\_test, y\_test))  knn\_weighted = KNeighborsClassifier(n\_neighbors=k, weights='distance')  knn\_weighted.fit(X\_train, y\_train)  accuracy\_weighted.append(knn\_weighted.score(X\_test, y\_test))  plt.plot(neighbors\_range, accuracy\_unweighted, label='Unweighted kNN')  plt.plot(neighbors\_range, accuracy\_weighted, label='Weighted kNN')  plt.xlabel('Number of Neighbors')  plt.ylabel('Accuracy')  plt.title('Accuracy vs Number of Neighbors for kNN')  plt.legend()  plt.show() |
| --- |

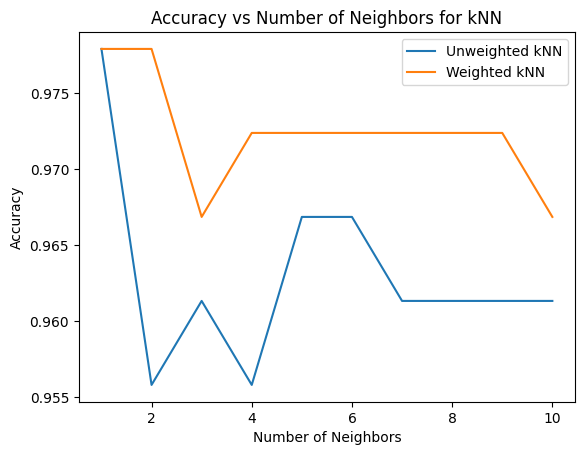


Рисунок 2.1 – зависимость точности от количества соседей для взвешенного и невзвешенного методов

Из рисунка 2.1 можно увидеть, что лучшая точность у обоих методов достигается при количестве соседей = 1.

* 1. Построим изображение с границами принятия решения(листинг 2.2, рис. 2.2).

Листинг 2.2 – Построение изображения с границами принятия решения.

| cmap\_light = ListedColormap(["blue", "red"])  fig, axes = plt.subplots(1, 2, figsize=(14, 6))  display\_unweighted = DecisionBoundaryDisplay.from\_estimator(  knn\_unweighted, X\_test, response\_method='predict',  xlabel='X1', ylabel='X2', ax=axes[0],  grid\_resolution=200, alpha = 0.3, cmap = cmap\_light)  axes[0].set\_title(f'kNN unweighted')  display\_unweighted.ax\_.scatter(X\_test[:,0], X\_test[:,1], c=colors\_test, edgecolors = 'black')  display\_weighted = DecisionBoundaryDisplay.from\_estimator(  knn\_weighted, X\_test, response\_method='predict',  xlabel='X1', ylabel='X2', ax=axes[1],  grid\_resolution=200, alpha = 0.3, cmap = cmap\_light)  display\_weighted.ax\_.scatter(X\_test[:,0], X\_test[:,1], c=colors\_test, edgecolors = 'black')  axes[1].set\_title(f'kNN weighted')  plt.show() |
| --- |

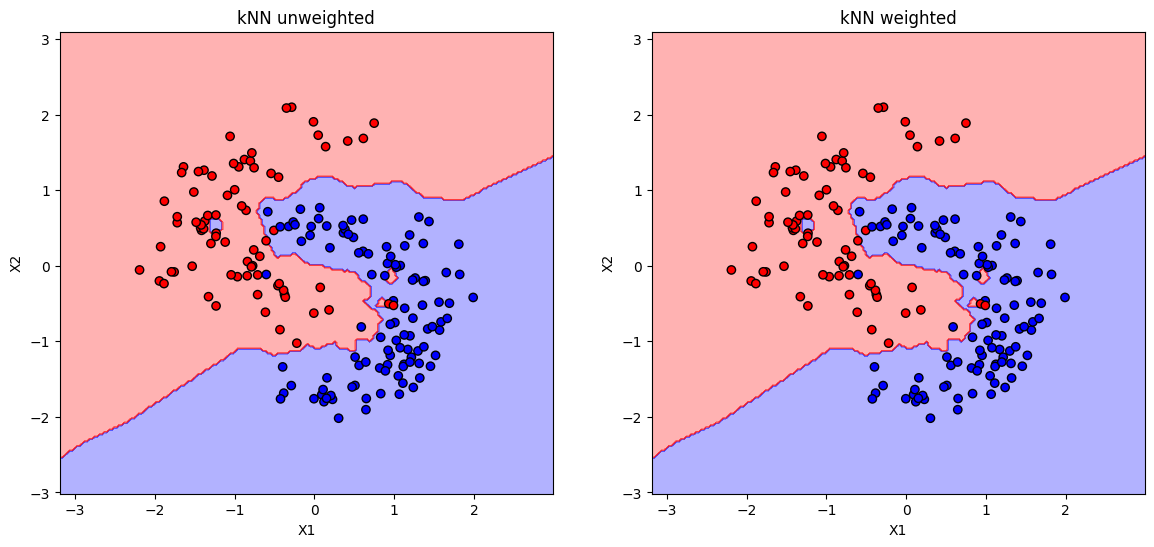


Рисунок 2.2 – изображение с границами принятия решения для взвешенного и невзвешенного методов.

Из рисунка 2.2 можно понять, что качество классификации обоих методов высокое, каждый из них четко отделяет границы каждого класса. Также можно заметить неустойчивость метода к выбросам, так как выделяются малые области противоположного класса внутри основного класса. Еще метод не учитывает некоторые наблюдения одного класса внутри другого класса.

2.4. Рассчитаем значения Precision, Recall, F1 для полученных результатов (листинг 2.4, табл. 2.4).

Листинг 2.4 – подсчет Precision, Recall, F1

| precision\_unweighted = precision\_score(y\_test, y\_pred\_unweighted)  recall\_unweighted = recall\_score(y\_test, y\_pred\_unweighted)  f1\_unweighted = f1\_score(y\_test, y\_pred\_unweighted)  precision\_weighted = precision\_score(y\_test, y\_pred\_weighted)  recall\_weighted = recall\_score(y\_test, y\_pred\_weighted)  f1\_weighted = f1\_score(y\_test, y\_pred\_weighted) |
| --- |

Таблица 2.4 – показатели Precision, Recall, F1 для обоих методов

|  | Невзвешенный | Взвешенный |
| --- | --- | --- |
| Precision | 0.9736842105263158 | 0.9736842105263158 |
| Recall | 0.9736842105263158 | 0.9736842105263158 |
| F1 | 0.9736842105263158 | 0.9736842105263158 |

Precision показывает сколько из истинных предсказаний являются реально истинными.

Recall показывает сколько реально истинных значений были правильно определены как истинные.

F1-мера – среднее гармоническое из Precision и Recall.

Из таблицы 2.4 можно сказать, что рассчитанные метрики имеют высокое значение в силу малого количества ошибочных предсказаний.

1. Логистическая регрессия
   1. Проведем классификацию методом логистической регрессии при различных параметрах: без регуляризации, с l1 регуляризацией, c l2 регуляризацией. Построим столбчатую диаграмму зависимости точности от наличия регуляризации.

Классификация и построение диаграммы указаны в листинге 3.1, результат на рис. 3.1

Листинг 3.1 – Классификация и построение диаграммы

| logreg\_no\_reg = LogisticRegression(penalty=None, solver='lbfgs', max\_iter=1000)  logreg\_l1 = LogisticRegression(penalty='l1', solver='liblinear', max\_iter=1000)  logreg\_l2 = LogisticRegression(penalty='l2', solver='lbfgs', max\_iter=1000)  logreg\_no\_reg.fit(X\_train, y\_train)  logreg\_l1.fit(X\_train, y\_train)  logreg\_l2.fit(X\_train, y\_train)  y\_pred\_no\_reg = logreg\_no\_reg.predict(X\_test)  y\_pred\_l1 = logreg\_l1.predict(X\_test)  y\_pred\_l2 = logreg\_l2.predict(X\_test)  accuracy\_no\_reg = accuracy\_score(y\_test, y\_pred\_no\_reg)  accuracy\_l1 = accuracy\_score(y\_test, y\_pred\_l1)  accuracy\_l2 = accuracy\_score(y\_test, y\_pred\_l2)  models = ['Без регуляризации', 'L1', 'L2']  accuracies = [accuracy\_no\_reg, accuracy\_l1, accuracy\_l2]  plt.bar(models, accuracies, color=['r', 'g', 'b'])  plt.xlabel('Регуляризация')  plt.ylabel('Точность')  plt.title('Точность от регуляризации')  plt.ylim([0, 1])  plt.show() |
| --- |

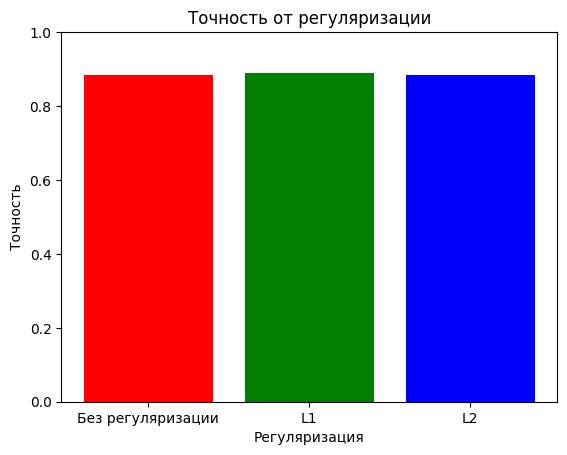


Рисунок 3.1 – диаграмма зависимости точности от типа регуляризации

Из рисунка 3.1 видно, что точность у всех типов почти одинаковая, но L1 показывает чуть более высокую точность. Далее будем использовать результат с L1.

* 1. Построим изображение с границами принятия решения (рис. 3.2)

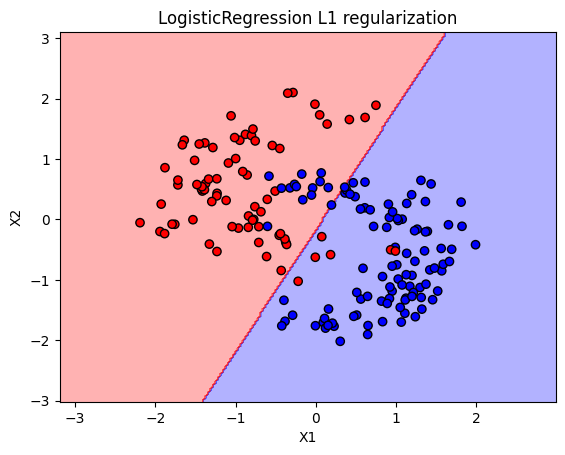
. 

Рисунок 3.2 – изображение с границами принятия решений для логистической регрессии с l1-регуляризацией

Из рисунка 3.2 видно, что логистическая регрессия линейно разделила данные нелинейной формы, что привело к попаданию наблюдений одного класса в другой. В данном случае качество классификации невысокое.

3.4. Рассчитаем значения Precision, Recall, F1 для полученных результатов (листинг 3.4, табл. 3.4).

Листинг 3.4 – подсчет Precision, Recall, F1

| precision\_logreg\_l1 = precision\_score(y\_test, y\_pred\_l1)  recall\_logreg\_l1 = recall\_score(y\_test, y\_pred\_l1)  f1\_logreg\_l1 = f1\_score(y\_test, y\_pred\_l1)  print("Precision, Recall и F1-score для LogisticRegression L1 regularization:")  print("Precision:", precision\_logreg\_l1)  print("Recall:", recall\_logreg\_l1)  print("F1-score:", f1\_logreg\_l1) |
| --- |

Таблица 3.4 – показатели Precision, Recall, F1 для логистической регрессии с L1-регуляризацией

|  | L1 |
| --- | --- |
| Precision | 0.8333333333333334 |
| Recall | 0.9210526315789473 |
| F1 | 0.875 |

Из таблицы 3.4 можно сказать, что, исходя из величины Precision, количество правильно определенных истинных значений не много, так как классификатор допустил достаточное количество ошибок. Но Recall показывает, что большинство действительно истинных значений были правильно определены как истинные.

1. Метод опорных векторов
   1. Проведем классификацию методом опорных векторов при различных параметрах ядра: “linear”, “poly”, “rbf”. Построим столбчатую диаграмму зависимости точности от вида ядра.

Для параметра poly используем степень 3, т.к. при переборе значений от 1 до 10 оно дало наилучший результат.

Классификация представлена в листинге 4.1, результат на рисунке 4.1. Код построения диаграммы такой же, как в прошлом пункте.

Листинг 4.1 – классификация

| svm\_linear = SVC(kernel='linear', probability=True)  svm\_poly = SVC(kernel='poly', degree=3, probability=True)  svm\_rbf = SVC(kernel='rbf', probability=True)  svm\_linear.fit(X\_train, y\_train)  svm\_poly.fit(X\_train, y\_train)  svm\_rbf.fit(X\_train, y\_train)  y\_pred\_linear = svm\_linear.predict(X\_test)  y\_pred\_poly = svm\_poly.predict(X\_test)  y\_pred\_rbf = svm\_rbf.predict(X\_test)  accuracy\_linear = accuracy\_score(y\_test, y\_pred\_linear)  accuracy\_poly = accuracy\_score(y\_test, y\_pred\_poly)  accuracy\_rbf = accuracy\_score(y\_test, y\_pred\_rbf) |
| --- |

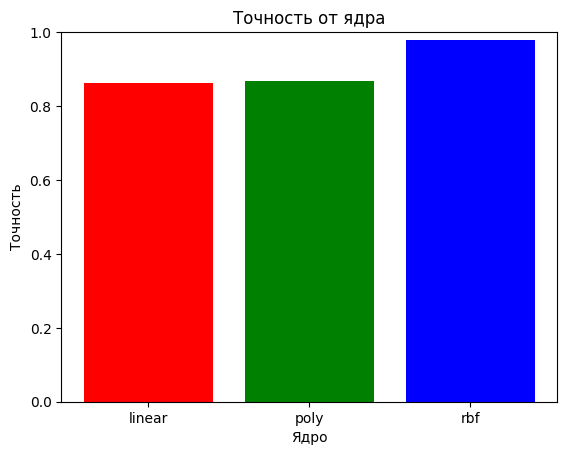


Рисунок 4.1 – диаграмма зависимости точности от типа ядра

Из рисунка 4.1 видно, что тип rbf показывает самую высокую точность. Далее будем использовать результат с rbf.

* 1. Построим изображение с границами принятия решения (рис. 4.2)

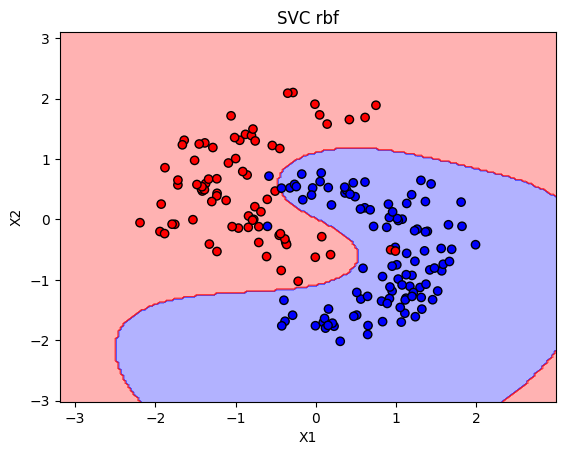
. 

Рисунок 4.2 – изображение с границами принятия решений для метода опорных векторов с rbf.

Из рисунка 4.2 видно, что метод попытался разделить данные в соответствии с их формой. Метод не способен обработать точки, лежащие внутри другого класса, однако, из-за их малого количества, их можно отнести к выбросам для окружающего их класса. Качество полученной классификации можно назвать хорошим.

4.4 Рассчитаем значения Precision, Recall, F1 для полученных результатов (листинг 4.4, табл. 4.4).

Листинг 4.4 – подсчет Precision, Recall, F1

| precision\_logreg\_l1 = precision\_score(y\_test, y\_pred\_l1)  recall\_logreg\_l1 = recall\_score(y\_test, y\_pred\_l1)  f1\_logreg\_l1 = f1\_score(y\_test, y\_pred\_l1)  print("Precision, Recall и F1-score для LogisticRegression L1 regularization:")  print("Precision:", precision\_logreg\_l1)  print("Recall:", recall\_logreg\_l1)  print("F1-score:", f1\_logreg\_l1) |
| --- |

Таблица 4.4 – показатели Precision, Recall, F1 для метода опорных векторов с параметром ядра rbf

|  | rbf |
| --- | --- |
| Precision | 0.9736842105263158 |
| Recall | 0.9736842105263158 |
| F1 | 0.9736842105263158 |

Из таблицы 4.4 можно сказать, что для каждого класса было дано мало ошибочных предсказаний, т.к. все метрики одинаковы и высоки.

1. Решающие деревья
   1. Проведем классификацию используя решающие деревья, подобрав параметры при которых получается лучшее обобщение.

Для нахождения лучшего набора параметров для классификации воспользуемся кросс-валидацией (листинг 5.1).

Листинг 5.1 – подбор лучших параметров для классификации.

| param\_grid = {  'max\_depth': [3, 5, 7, None],  'max\_leaf\_nodes': [None, 5, 10, 20],  'criterion': ['gini', 'entropy'],  "min\_samples\_split": [2, 3, 4, 5]  }  dt = DecisionTreeClassifier()  grid\_search = GridSearchCV(dt, param\_grid, cv=5)  grid\_search.fit(X\_train, y\_train)  best\_params = grid\_search.best\_params\_  best\_dt = grid\_search.best\_estimator\_  y\_pred\_dt = best\_dt.predict(X\_test)  print("Best parameters:", best\_params) |
| --- |

В результате кросс-валидации мы получили следующие оптимальные параметры:

* макс. глубина – 7
* макс. кол-во листьев – не установлено
* метрика загрязнения – энтропия
* мин. кол-во наблюдений для разбиения – 2
  1. Построим изображение с границами принятия решения (рис. 5.2)

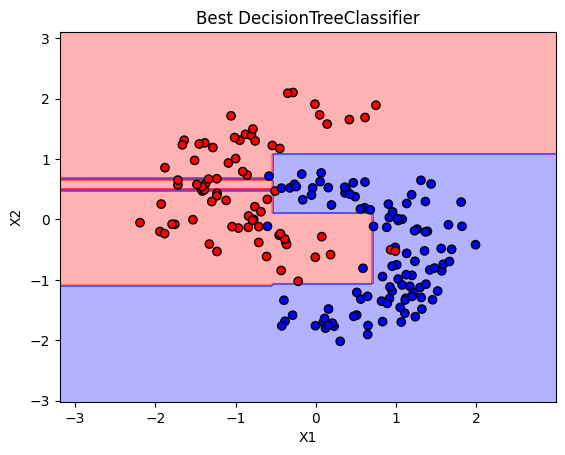
. 

Рисунок 5.2 – изображение с границами принятия решений для метода решающего дерева с оптимальными параметрами.

Из рисунка 5.2 видно, что метод попытался разделить данные в соответствии с их формой. Метод не обработал точки, лежащие внутри другого класса, однако, из-за их малого количества, их можно отнести к выбросам для окружающего их класса. Качество полученной классификации можно назвать хорошим.

5.4. Рассчитаем значения Precision, Recall, F1 для полученных результатов (листинг 5.4, табл. 5.4).

Листинг 5.4 – подсчет Precision, Recall, F1

| precision\_logreg\_l1 = precision\_score(y\_test, y\_pred\_l1)  recall\_logreg\_l1 = recall\_score(y\_test, y\_pred\_l1)  f1\_logreg\_l1 = f1\_score(y\_test, y\_pred\_l1)  print("Precision, Recall и F1-score для LogisticRegression L1 regularization:")  print("Precision:", precision\_logreg\_l1)  print("Recall:", recall\_logreg\_l1)  print("F1-score:", f1\_logreg\_l1) |
| --- |

Таблица 5.4 – показатели Precision, Recall, F1 для метода решающего дерева с оптимальными параметрами

|  | Решающее дерево |
| --- | --- |
| Precision | 0.9230769230769231 |
| Recall | 0.9473684210526315 |
| F1 | 0.935064935064935 |

Из таблицы 5.4 можно сказать, что количество ошибочных определений невелико, т.к. метрики достаточно высокие.

5.6. Визуализируем полученное дерево решений (листинг 5.6, рис. 5.6).

Листинг 5.6 – визуализация дерева решений

| fig, ax = plt.subplots(1, 1, figsize=(40, 20))  plot\_tree(best\_dt, filled = True, ax=ax) |
| --- |

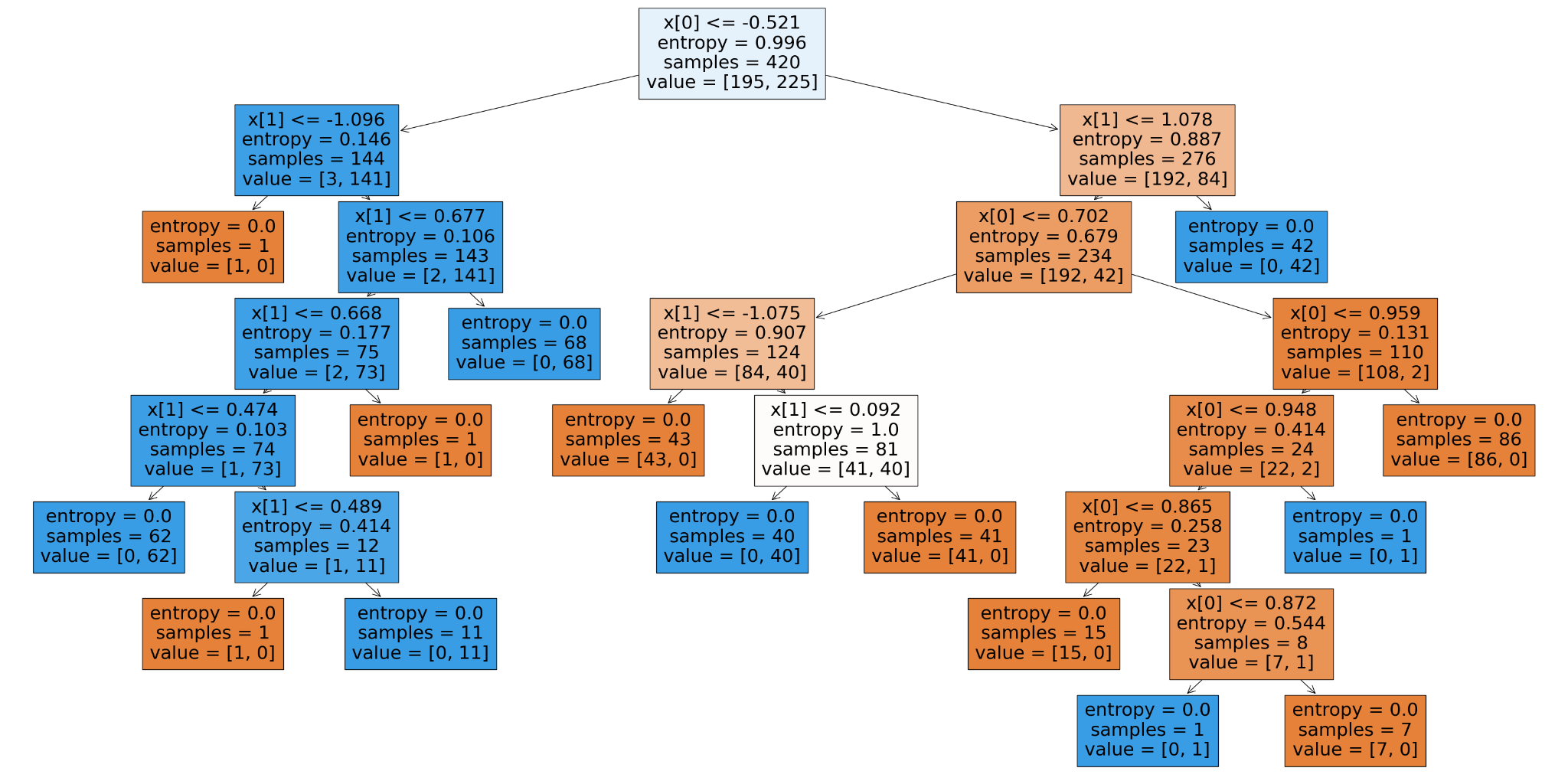


Рисунок 5.6 – визуализация дерева решений.

Из рисунка 5.6 мы видим, что в каждом узле правило распределяет наблюдения по классам, основываясь на первой или второй переменной из набора данных. Оранжевый класс – удовлетворяет условию 1 переменной, синий – 2. Также мы видим в обоих классах присутствие листов с одним наблюдением, что отсылает нас к точкам на рис. 5.2, которые лежат не в зоне своего класса. Этим же можно объяснить и появившиеся 2 полосы синего класса в красном классе. Также можно сказать, что дерево решений практически соответствует рис. 5.2 с границами принятия решения.

1. Выбор классификатора
   1. Построим таблицу с метриками Precision, Recall для полученных результатов каждым классификатором (табл. 6.1).

Таблица 6.1 – метрики по каждому классификатору

|  | **Precision** | **Recall** |
| --- | --- | --- |
| kNN невзвешенный | 0.973684 | 0.973684 |
| kNN взвешенный | 0.973684 | 0.973684 |
| Логистическая регрессия с L1-регуляризацией | 0.833333 | 0.921053 |
| Опорные вектора с rbf | 0.973684 | 0.973684 |
| Дерево решений | 0.923077 | 0.947368 |

* 1. Из таблицы 6.1 видно, что наилучшими классификаторами для набора данных lab4\_6 являются оба варианта kNN и метод опорных векторов с rbf, т.к. их метрики являются самыми высокими среди полученных.

Невзвешенный kNN хорошо отработает для небольшого количества наблюдений, которых окажется недостаточно для более сложных моделей.

Взвешенный kNN хорошо отработает для близких наблюдений с различающейся значимостью для классификации

Логистическая регрессия с L1 хорошо отработает в случае большого количества признаков, а также если нужно не допустить переобучения модели.

Метод опорных векторов с rbf хорошо отработает в случае сложной структуры данных и большого количества признаков.

Решающее дерево хорошо отработает в случае необходимости визуализации, а также при нелинейной зависимости между признаками и целевой переменной.

**Вывод.**

Были изучены следующие методы классификации данных:

* kNN
* Логистическая регрессия
* Метод опорных векторов
* Решающее дерево

Для каждого из методов были определены лучшие параметры для данного набора данных, построены изображения с границами принятия решения, рассчитаны метрики, по которым можно определить качество классификации.

Для метода решающего дерева была также сделана визуализация в виде графа.

Для данного набора данных были выбраны лучшие методы классификации.