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 5

Студент гр. 1303 _____ Самохин К. А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Изучить работу разных методов классификации. Рассмотреть работу методов при различных параметрах и выбрать метод классификации, наилучшим образом подходящий под исходный набор данных.

Задание.

1. Загрузка данных.
   1.1. Загрузите данные вашего варианта. Учтите, что метки классов являются текстом.
   1.2. Визуализируйте данные при помощи диаграммы рассеяния с выделением различных классов ней.
   1.3. Оцените сбалансированность классов.
   1.4. Проведите предобработку данных при необходимости.
   1.5. Разделите выборку на обучающую и тестовую. На обучающей проводите обучение модели, на тестовой расчет значений метрик. Оценка модели регрессии.
2. kNN.
   2.1. Проведите классификацию методом k ближайших соседей, подобрав параметры: количество соседей, необходимость взвешивания. Постройте графики зависимости точности (*Accuracy*) в зависимости от количества соседей (по 1-й линии для взвешенного и не взвешенного метода).
   2.2. Постройте изображение с границами принятия решения ответив на них точки классов разным цветом. Сделайте выводы о качестве классификации на основе полученного изображения.
   2.3. Постройте таблицу ошибок для полученных результатов. Сделайте выводы о классификации на основе таблицы ошибок.
   2.4. Рассчитайте значения *Precision*, *Recall*, *F1* для полученных результатов. Сопоставьте с таблицей ошибок.

2.5. Постройте изображение *ROC*-кривой и рассчитайте значение *AUC* для полученных результатов. Сделайте выводы о классификации по полученному изображение и значению *AUC*.

3. Логистическая регрессия.
   3.1. Проведите классификацию методом логистической регрессии при различных параметрах: без регуляризации, с *l1* регуляризацией, с *l2* регуляризацией. Постройте столбчатую диаграммузависимости точности (*Accuracy*) от наличия регуляризации. Дальнейшие пункты 3.* выполняйте для лучшего параметра.
   3.2. Постройте изображение с границами принятия решения ответив на них точки классов разным цветом. Сделайте выводы о качестве классификации на основе полученного изображения.
   3.3. Постройте таблицу ошибок для полученных результатов. Сделайте выводы о классификации на основе таблицы ошибок.
   3.4. Рассчитайте значения *Precision*, *Recall*, *F1* для полученных результатов. Сопоставьте с таблицей ошибок.
   3.5. 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.
   4.1. 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.
   4.2. Постройте изображение с границами принятия решения ответив на них точки классов разным цветом. Сделайте выводы о качестве классификации на основе полученного изображения.
   4.3. Постройте таблицу ошибок для полученных результатов. Сделайте выводы о классификации на основе таблицы ошибок.

4.4. Рассчитайте значения *Precision*, *Recall*, *F1* для полученных результатов. Сопоставьте с таблицей ошибок.

4.5. Постройте изображение *ROC*-кривой и рассчитайте значение *AUC* для полученных результатов. Сделайте выводы о классификации по полученному изображение и значению *AUC*.

5. Решающие деревья.

5.1. Проведите классификацию используя решающие деревья, подобрав параметры при которых получается лучшее обобщение (максимальная глубина / максимальное количество листьев / метрика загрязнения и т.д.).

5.2. Постройте изображение с границами принятия решения ответив на них точки классов разным цветом. Сделайте выводы о качестве классификации на основе полученного изображения.

5.3. Постройте таблицу ошибок для полученных результатов. Сделайте выводы о классификации на основе таблицы ошибок.

5.4. Рассчитайте значения *Precision*, *Recall*, *F1* для полученных результатов. Сопоставьте с таблицей ошибок.

5.5. Постройте изображение *ROC*-кривой и рассчитайте значение *AUC* для полученных результатов. Сделайте выводы о классификации по полученному изображение и значению *AUC*.

5.6. Визуализируйте полученное дерево решений. Сделайте выводы о правилах в узлах дерева. Сопоставьте их с полученными границами принятия решений.

6. Выбор классификатора.

6.1. Постройте таблицу с метриками *Precision*, *Recall*, *AUC* для полученных результатов каждым классификатором.

6.2. Сделайте выводы о том, какие классификаторы лучше всего подходят для вашего набора данных и в каких случаях.

Выполнение работы.

1. Загрузка данных.

1.1. Загрузим данные из файла *lab4_2.csv*, используя метод *read_csv* (см. листинг 1.1.1). Корректность загрузки датасета проверим, вызвав метод *head* (см. листинг 1.1.2). Результат работы метода представлен в таблице 1.1.1.

Листинг 1.1.1 – Считывание датасета из файла lab4_2.csv.

```
df = pd.read_csv('lab4_2.csv')
```

Листинг 1.1.2 – Вызов метода *head.*

```
df.head()
```

Таблица 1.1.1 – Результат работы метода *head.*

	X1	X2	Class
0	-0.825533	1.177218	I
1	-1.424662	1.262654	I
2	-3.309480	1.062984	I
3	2.375619	1.035644	I
4	-0.202036	1.451284	I

1.2. Визуализируем данные при помощи диаграммы рассеяния с выделением наблюдений разных классов (см. листинг 1.2.1). Полученная диаграмма представлена на рисунке 1.2.1.

Листинг 1.2.1 – Рисование диаграмм рассеяния.

```
g = sb.PairGrid(data=df, hue="Class", palette="tab10")
g.map(sb.scatterplot)
g.add_legend()
```

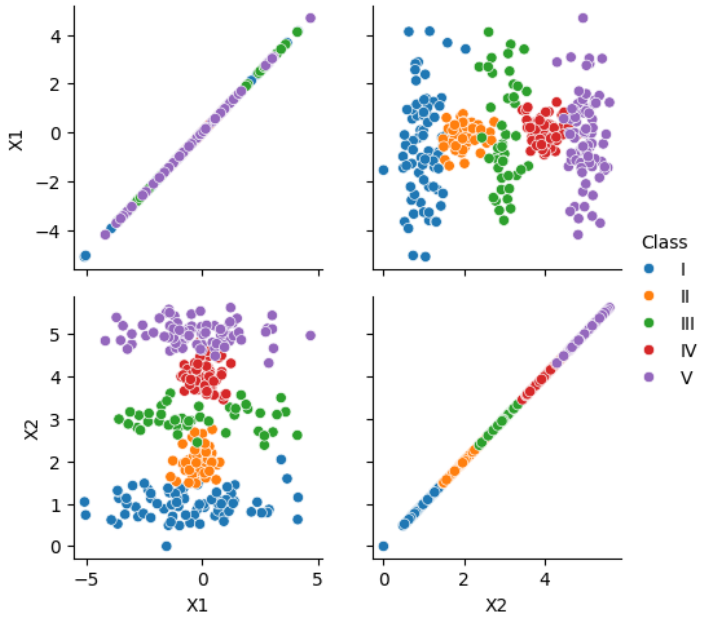

Рисунок 1.2.1 – Диаграммы рассеяния.

1.3. Из рисунка 1.2.1 трудно сделать выводы о сбалансированности классов, поэтому принято решение программно подсчитать число наблюдений каждого класса (см. листинг 1.3.1). Результат подсчётов см. в листинге 1.3.2.

Листинг 1.3.1 – Вычисление количества наблюдений каждого класса.

```
class_counts = df['Class'].value_counts()
class_counts
```

Листинг 1.3.2 – Результат подсчёта наблюдений.

```
Class
I      77
V      77
II     55
IV     55
III    44
```

Как можно заметить, наблюдения распределены по классам не равномерно, число наблюдений классов *“I”* и *“V”* значительно превосходит число наблюдений классов *“II”*, *“IV”* и *“III”*.

1.4. Проведём предобработку данных. Закодируем метки классов (см. листинг 1.4.1).

Листинг 1.4.1 – Кодирование меток классов.

```
le = LabelEncoder()
df["Class"] = le.fit_transform(df["Class"])
```

1.5. Разделим выборку на тестовую и обучающую, используя метод *train_test_split* из библиотеки *Scikit-learn* (см. листинг 1.5.1).

Листинг 1.5.1 – Разбиение выборки на тестовую и обучающую.

```
df_train, df_test = train_test_split(df, test_size = 0.3)

x_train = df_train[["X1", "X2"]]
x_test = df_test[["X1", "X2"]]
```

2. kNN.

2.1 Проведём классификацию методом kNN, подобрав параметры: количество соседей и веса (см. листинг 2.1.1). Построим график зависимости точности (*Accuracy*) от количества соседей (см. листинг 2.1.2). Полученный график см. на рисунке 2.1.1.

Листинг 2.1.1 – Подбор параметров для классификации.

```
accuracy_distance = []
accuracy_uniform = []

for k in range(1, 20):
      knn_uniform = KNeighborsClassifier(n_neighbors=k)
      knn_uniform.fit(x_train, df_train["Class"])
      accuracy_uniform.append(knn_uniform.score(x_test,
      df_test["Class"]))

      knn_distance = KNeighborsClassifier(n_neighbors=k,
      weights='distance')
      knn_distance.fit(x_train, df_train["Class"])
```

```
11          accuracy_distance.append(knn_distance.score(x_test,
            df_test["Class"]))
```

В листинге 2.1.1 для подбора оптимальных параметров используется цикл, в котором осуществляется перебор значений параметра *n_neighbors*. Важно отметить, что на каждой итерации цикла точность вычисляется сразу для двух значений параметра *weights*, результат вычислений помещается в 2 разных списка, что в дальнейшем упрощает рисование графика.

Листинг 2.1.2 – Рисование графика зависимости точности от количества соседей

```
1     plt.plot(range(1, 20), accuracy_uniform, label='unoform')
2     plt.plot(range(1, 20), accuracy_distance, label='distance')
3     plt.ylabel("Точность")
4     plt.xlabel("Количество соседей")
5     plt.legend()
6     plt.show()
```

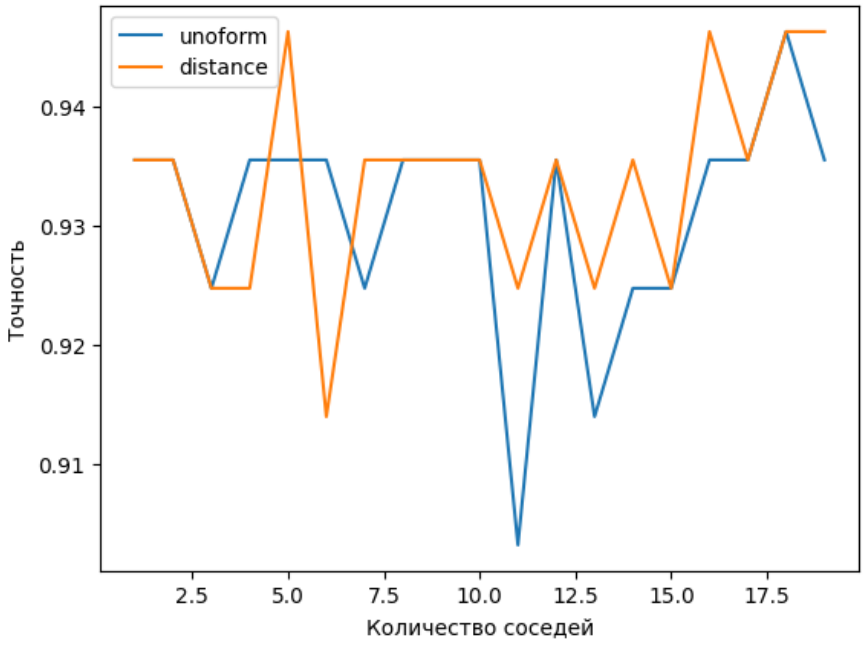

Рисунок 2.1.1 – График зависимости точности от количества соседей.

На рисунке 2.1.1 видно, что лучшая точность достигается при выборе 5 ближайших соседей со взвешиванием.

2.2. Построим изображение с границами принятия решения, отметив на них точки классов разным цветом (см. листинг 2.2.1). Полученную диаграмму см. на рисунке 2.2.1.

Листинг 2.2.1 – Рисование границ принятия решений с точками классов.

```
knn = KNeighborsClassifier(n_neighbors=5, weights="distance")
knn.fit(X=x_train, y=df_train["Class"])
test_pred = knn.predict(X=x_test)

disp = DecisionBoundaryDisplay.from_estimator(estimator=knn,
X=df[["X1", "X2"]], xlabel="X1", ylabel="X2",
grid_resolution=200, alpha = 0.7)
scatt = disp.ax_.scatter( x_test["X1"].to_numpy(),
x_test["X2"].to_numpy(), c=test_pred, edgecolors = "black")
handles, labels = scatt.legend_elements()
plt.legend(handles=handles, labels=[str(i) for i in
knn.classes_])
plt.show()
```

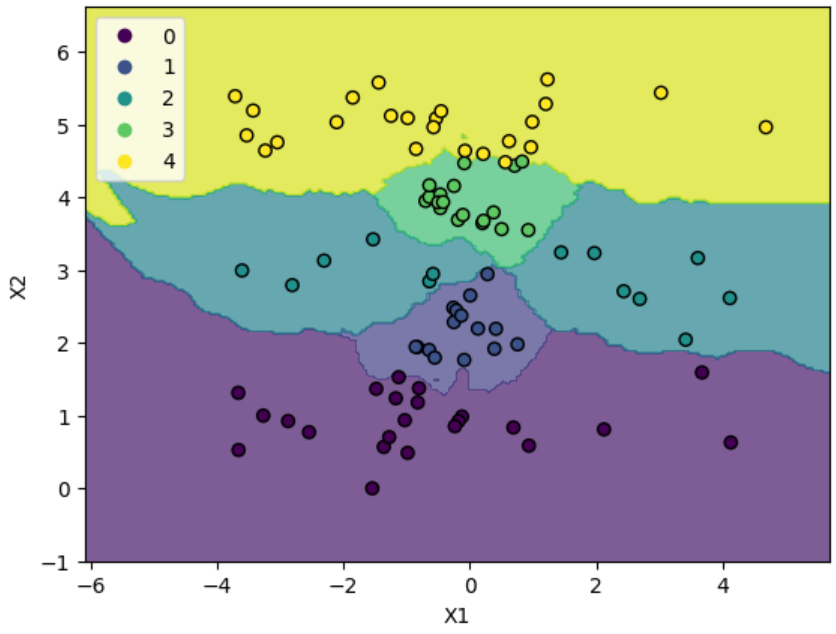

Рисунок 2.2.1 – Иллюстрация границ классов.

На рисунке 2.2.1 видно, что классификация прошла успешно. Классы стали более различимы. Большая часть точек находятся в своих областях, однако есть

единичные случаи, когда точки располагаются на границе двух областей, что затрудняет определение их принадлежности к определённому классу.

2.4. Рассчитаем значения *Precision, Recall, F1* для полученных результатов (см. листинг 2.4.1). Результат вычислений см. в листинге 2.4.2.

Листинг 2.4.1 – Вычисление метрик.

```
1    prfs=precision_recall_fscore_support( y_true=df_test["Class"].
     to_numpy() , y_pred=test_pred, labels=knn.classes_)
2
3    print(f"Precision:\t{prfs[0]}")
4    print(f"Recall:\t{prfs[1]}")
5    print(f"F1-score:\t{prfs[2]}")
```

Листинг 2.3.2 – Значения метрик.

```
Precision: [0.95652174 0.86666667 0.92307692 1.         0.95833333]
Recall:    [0.95652174 0.92857143 0.85714286 0.94736842 1.        ]
F1-score:  [0.95652174 0.89655172 0.88888889 0.97297297 0.9787234 ]
```

Из листинга 2.3.2 можно сделать вывод, что значение метрики *Precision* для всех классов принимает очень большое значение (для класса 3 значение вовсе равно 100%), это говорит об очень высокой вероятности успешного прогнозирования истинного результата. Значение метрики *Recall* также очень большое для всех классов (для класса 4 равно 100%), это свидетельствует о высокой вероятности прогнозирования истинного результата на основе истинных значений. Метрика *F1* также имеет очень большое значение, что позволяет судить о хорошей сбалансированности данных и высоком качестве модели.

3. Логическая регрессия.

3.1. Проведём классификацию методом регрессии, подобрав значение параметра *penalty* (без регуляризации, с l1 регуляризацией, с l2 регуляризацией) (см. листинг 3.1.1). Построим график зависимости точности (*Accuracy*) от регуляризации (см. листинг 3.1.2). Полученный график см. на рисунке 3.1.1.

Листинг 3.1.1 – Подбор параметров для классификации.

```
1    penalties = [None, "l1", "l2"]
2    accuracy = []
3
4    for penalty in penalties:
5         logreg = LogisticRegression(penalty=penalty, solver='saga',
          max_iter=2000)
6         logreg.fit(X=x_train, y=df_train["Class"])
7         test_pred = logreg.predict(x_test)
8         acc = accuracy_score( y_true=df_test["Class"].to_numpy(),
          y_pred=test_pred)
9         accuracy.append(acc)
```

В листинге 3.1.1 для подбора оптимальных параметров используется цикл, в котором осуществляется перебор значений параметра *penalty*. Результат вычисления точности для всех значений параметра записывается в массив *accuracy*.

Листинг 3.1.2 – Рисование гистограммы зависимости точности от параметра *penalty.*

```
1    sb.barplot(x=[str(pen) for pen in penalties], y=accuracy)
2    plt.ylim(min(accuracy) - 0.01, max(accuracy) + 0.01)
3    plt.ylabel("Точность")
4    plt.xlabel("Регуляризация")
5    plt.show()
```

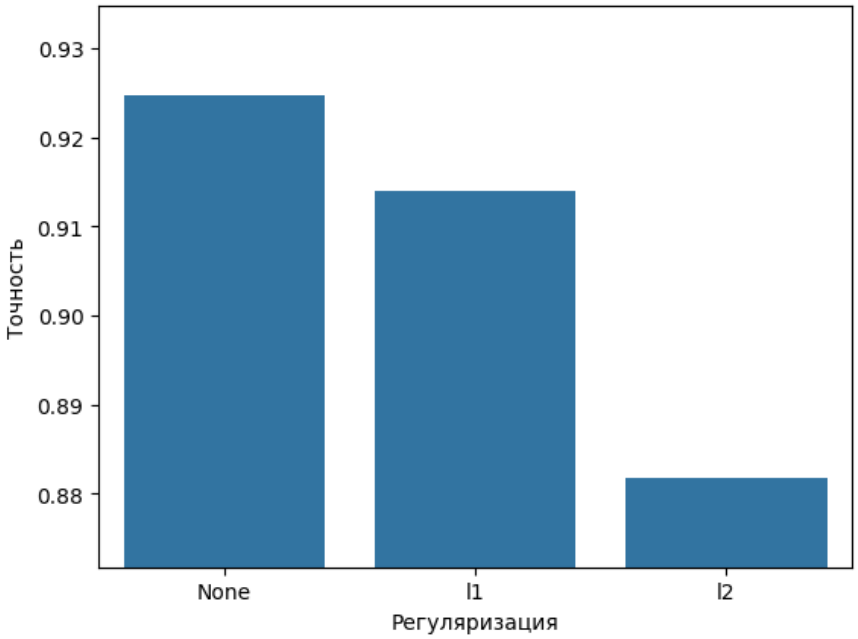

Рисунок 3.1.1 – Гистограмма зависимости точности от параметра *penalty*.

На рисунке 3.1.1 видно, что наилучшая точность достигается без регуляризации.

3.2. Построим изображение с границами принятия решения, отметив на них точки классов разным цветом (см. листинг 3.2.1). Полученную диаграмму см. на рисунке 3.2.1.

Листинг 3.2.1 – Рисование границ принятия решений с точками классов.

```
1    logreg = LogisticRegression(penalty=None, solver='saga',
     max_iter=2000)
2    logreg.fit(X=x_train, y=df_train["Class"])
3    test_pred = logreg.predict(X=x_test)
4
5    disp = DecisionBoundaryDisplay.from_estimator(estimator=logreg,
     X=df[["X1", "X2"]], xlabel="X1", ylabel="X2",
     grid_resolution=200, alpha = 0.7)
6    scatt = disp.ax_.scatter( x_test["X1"].to_numpy(),
     x_test["X2"].to_numpy(), c=test_pred, edgecolors = "black")
7    handles, labels = scatt.legend_elements()
8    plt.legend(handles=handles, labels=[str(i) for i in
     logreg.classes_])
9    plt.show()
```

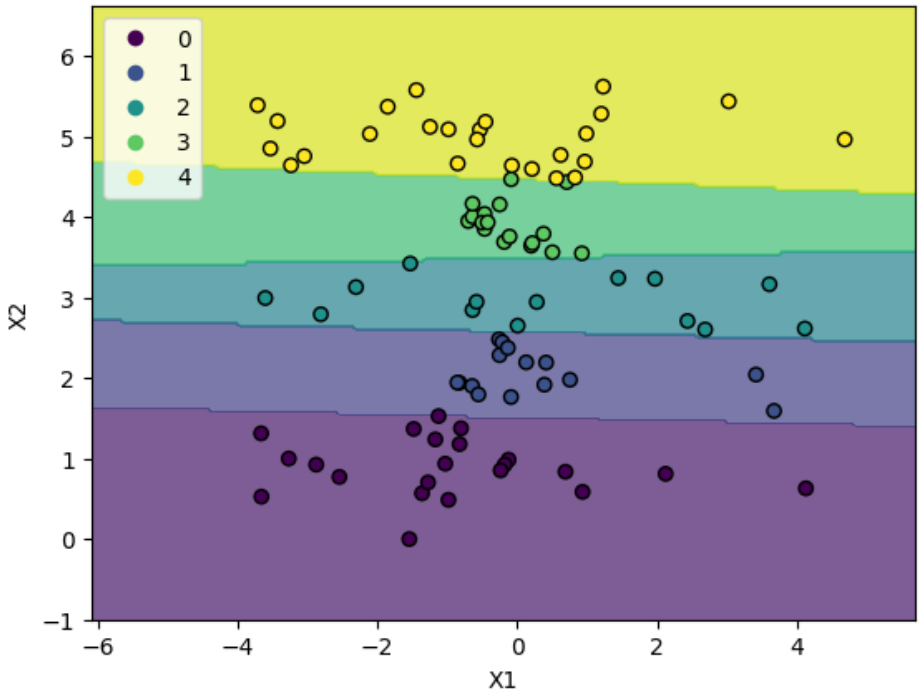

Рисунок 3.2.1 – Иллюстрация границ классов.

На рисунке 3.2.1 видно, что классификация прошла успешно. Классы стали более различимы. Почти все точки находятся в своих областях. Однако, есть несколько наблюдений, находящихся на границе раздела областей классов, из-за чего они могут быть неправильно идентифицированы.

3.4. Рассчитаем значения *Precision*, *Recall*, *F1* для полученных результатов (см. листинг 3.4.1). Результат вычислений см. в листинге 3.4.2.

Листинг 3.4.1 – Вычисление метрик.

```
prfs = precision_recall_fscore_support( y_true=df_test["Class"].
to_numpy(), y_pred=test_pred, labels=logreg.classes_)

print(f"Precision:\t{prfs[0]}")
print(f"Recall:\t{prfs[1]}")
print(f"F1-score:\t{prfs[2]}")
```

Листинг 3.4.2 – Значения метрик.

```
Precision: [0.95454545 0.8        0.92857143 1.         0.92       ]
Recall:    [0.91304348 0.85714286 0.92857143 0.89473684 1.         ]
F1-score:  [0.93333333 0.82758621 0.92857143 0.94444444 0.95833333]
```

Из листинга 3.4.2 можно сделать вывод, что значение метрики *Precision* для большинства классов принимает очень большое значение (для класса 3 значение вовсе равно 100%), однако для класса 1 значение метрики равно всего 80%, это говорит о средней вероятности успешного прогнозирования истинного результата. Значение метрики *Recall* также очень большое для большинства классов, однако для класса 1 оно принимает наименьшее значение – 85%, это свидетельствует о хорошей вероятности прогнозирования истинного результата на основе истинных значений. Метрика *F1* также имеет очень большое значение (значение для класса 1 снова меньше значений для других классов – 82%), что позволяет судить о хорошей сбалансированности данных и высоком качестве модели.

4. Метод опорных векторов.

4.1. Проведём классификацию методом опорных векторов, подобрав значение параметра *kernel* (linear, poly со степенью, rbf) (см. листинг 4.1.1). Построим график зависимости точности (*Accuracy*) от выбранного ядра (см. листинг 4.1.2). Полученный график см. на рисунке 4.1.1.

Листинг 4.1.1 – Подбор параметров для классификации.

```
1     kernels = ["linear", "poly", "rbf"]
2     accuracy = []
3     degree = 0
4
5     for kernel in kernels:
6           if kernel == 'poly':
7                max_acc = 0
8                for n in range(1,11):
9                     svm = SVC(kernel=kernel, degree=n, probability
                      =True)
10                    svm.fit(X=x_train, y=df_train["Class"])
11                    test_pred = svm.predict(x_test)
12                    acc = accuracy_score( y_true =df_test["Class"].
                      to_numpy(), y_pred=test_pred)
```

```
13                    if acc > max_acc:
14                          max_acc = acc
15                          Degree = n
16                    accuracy.append(max_acc)
17          else:
18                svm = SVC(kernel=kernel, probability=True)
19                svm.fit(X=x_train, y=df_train["Class"])
20                test_pred = svm.predict(x_test)
21                acc = accuracy_score( y_true=df_test["Class"].
                  to_numpy(), y_pred=test_pred)
22                accuracy.append(acc)
```

В листинге 4.1.1 для подбора оптимальных параметров используется цикл, в котором перебираются значения параметра *kernel*. В случае, если текущее проверяемое значение этого параметра равно *“poly”*, начинается перебор значений параметра *degree* – степени. Результаты точности при каждой классификации записываются в массив *accuracy*. Для ядра *poly* записывается только максимальное значение точности.

Листинг 4.1.2 – Рисование гистограммы зависимости точности от параметра *kernel.*

```
1     sb.barplot(x=kernels, y=accuracy)
2     plt.ylim(min(accuracy) - 0.01, max(accuracy) + 0.01)
3     plt.ylabel("Точность")
4     plt.xlabel("Ядро")
5     plt.show()
```

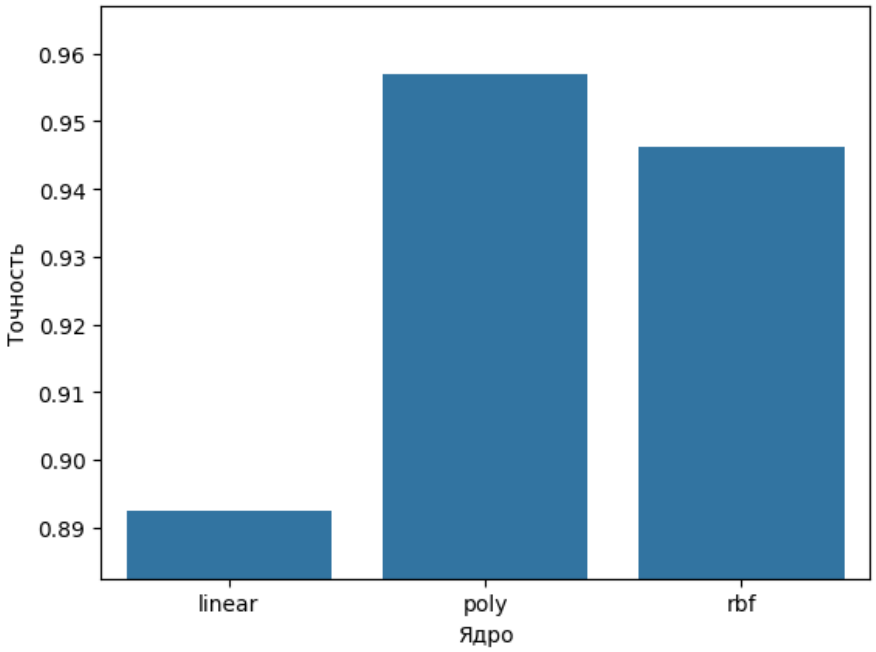

Рисунок 4.1.1 – Гистограмма зависимости точности от параметра *kernel.*

На рисунке 4.1.1 видно, что наилучшая точность достигается при выборе ядра *poly*. Было решено использовать вариант с ядром *poly* и значением степени 2.

4.2. Построим изображение с границами принятия решения, отметив на них точки классов разным цветом (см. листинг 4.2.1). Полученную диаграмму см. на рисунке 4.2.1.

Листинг 4.2.1 – Рисование границ принятия решений с точками классов.

```
svm = SVC(kernel='poly', degree=2, probability=True)
svm.fit(X=x_train, y=df_train["Class"])
test_pred = svm.predict(x_test)

disp = DecisionBoundaryDisplay.from_estimator(estimator=svm, X=df[["X1", "X2"]], xlabel="X1", ylabel="X2", grid_resolution=200, alpha = 0.7)
scatt = disp.ax_.scatter( x_test["X1"].to_numpy(), x_test["X2"].to_numpy(), c=test_pred, edgecolors = "black")
handles, labels = scatt.legend_elements()
plt.legend(handles=handles, labels=[str(i) for i in svm.classes_])
plt.show()
```

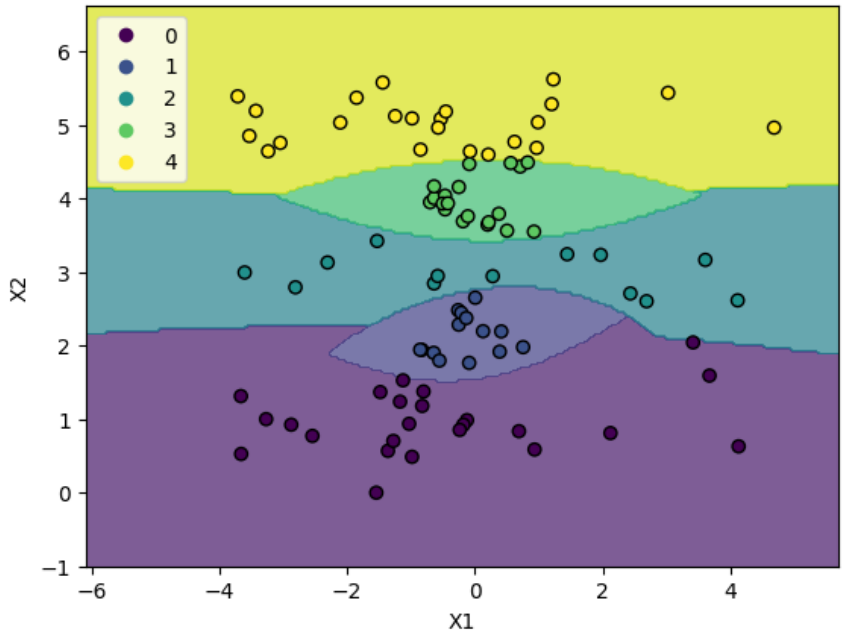

Рисунок 4.2.1 – Иллюстрация границ классов.

На рисунке 4.2.1 видно, что классификация прошла успешно. Классы стали более различимы. Все точки находятся в своих областях. Криволинейная форма границ классов позволяет более точно определять принадлежность точек к конкретному классу.

4.4. Рассчитаем значения *Precision*, *Recall*, *F1* для полученных результатов (см. листинг 4.4.1). Результат вычислений см. в листинге 4.4.2.

Листинг 4.4.1 – Вычисление метрик.

```
prfs = precision_recall_fscore_support( y_true=df_test["Class"].
to_numpy(), y_pred=test_pred, labels=svm.classes_)

print(f"Precision:\t{prfs[0]}")
print(f"Recall:\t{prfs[1]}")
print(f"F1-score:\t{prfs[2]}")
```

Листинг 4.4.2 – Значения метрик.

```
Precision: [0.95833333 0.92857143 1.         0.94736842 0.95652174]
Recall:    [1.         0.92857143 0.92857143 0.94736842 0.95652174]
F1-score:  [0.9787234  0.92857143 0.96296296 0.94736842 0.95652174]
```

Из листинга 4.4.2 можно сделать вывод, что значение метрики *Precision* для всех классов принимает значение свыше 90% (для класса 2 значение вовсе равно 100%), это говорит об очень высокой вероятности успешного прогнозирования истинного результата. Значение метрики *Recall* также очень большое для всех классов (для класса 0 равно 100%), это свидетельствует об очень хорошей вероятности прогнозирования истинного результата на основе истинных значений. Метрика *F1* также имеет очень большое значение, что позволяет судить о хорошей сбалансированности данных и высоком качестве модели.

5. Решающие деревья.

5.1. Проведём классификацию методом решающих деревьев, подобрав параметры *max_depth, max_leaf_nodes, criterion* и *min_samples_split* (см. листинг 5.1.1).

Листинг 5.1.1 – Подбор параметров.

```
best_params = {}
max_acc = 0
criterions = ["gini", "entropy", "log_loss"]

for criterion in criterions:
     for depth in range(1, 11):
          for leaf_nodes in range(2, 6):
               for min_ss in range(2, 6):
                    dtc = DecisionTreeClassifier
                    (criterion=criterion, max_depth=depth,
                    max_leaf_nodes=leaf_nodes,
                    min_samples_split=min_ss)
                    dtc.fit(X=x_train, y=df_train["Class"])
                    zero_count = np.count_nonzero
                    (dtc.tree_.impurity == 0)
                    test_pred = dtc.predict(x_test)
                    acc = accuracy_score( y_true=
                    df_test["Class"].to_numpy(),
                    y_pred=test_pred)
                    if acc > max_acc:
```

```
15                                  max_acc = acc
16                                  best_params["max_depth"] = depth
17                                  best_params["max_leaf_nodes"] =
                                    leaf_nodes
18                                  best_params["criterion"] = criterion
19                                  best_params["min_samples_split"] =
                                    min_ss
```

В листинге 5.1.1 происходит перебор значений четырёх параметров, граничные значения подобраны таким образом, чтобы не было переобучения. Для каждого набора параметров вычисляется точность, и в конце перебора в словаре *best_params* остаются параметры, при которых была достигнута наибольшая точность.

5.2. Построим изображение с границами принятия решения, отметив на них точки классов разным цветом (см. листинг 5.2.1). Полученную диаграмму см. на рисунке 5.2.1.

Листинг 5.2.1 – Рисование границ принятия решений с точками классов.

```
1    dtc = DecisionTreeClassifier(criterion=best_params["criterion"],
     max_depth=best_params["max_depth"],
     max_leaf_nodes=best_params["max_leaf_nodes"],
     min_samples_split=best_params["min_samples_split"])
2    dtc.fit(X=x_train, y=df_train["Class"])
3    test_pred = dtc.predict(x_test)
4
5    disp = DecisionBoundaryDisplay.from_estimator(estimator=dtc,
     X=df[["X1", "X2"]], xlabel="X1", ylabel="X2",
     grid_resolution=200, alpha = 0.7)
6    scatt = disp.ax_.scatter( x_test["X1"].to_numpy(),
     x_test["X2"].to_numpy(), c=test_pred, edgecolors = "black")
7    handles, labels = scatt.legend_elements()
8    plt.legend(handles=handles, labels=[str(i) for i in
     dtc.classes_])
9    plt.show()
```

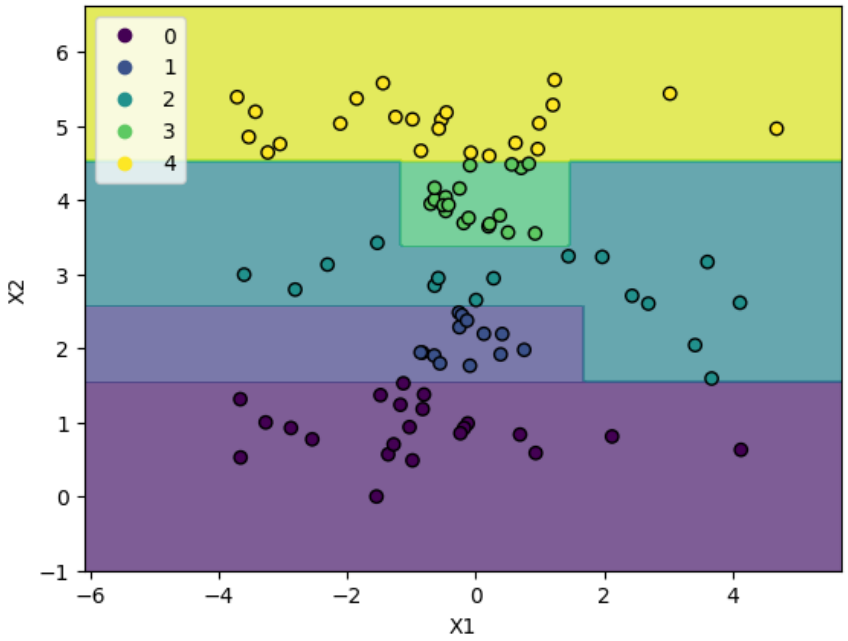

Рисунок 5.2.1 – Иллюстрация границ классов.

На рисунке 5.2.1 видно, что классификация прошла успешно. Классы стали более различимы. Почти все точки находятся в своих областях. Прямоугольная форма границ классов позволяет более точно определять принадлежность точек к конкретному классу. Однако, есть несколько наблюдений, находящихся на границе раздела областей классов, из-за чего они могут быть неправильно идентифицированы.

5.4. Рассчитаем значения *Precision*, *Recall*, *F1* для полученных результатов (см. листинг 5.4.1). Результат вычислений см. в листинге 5.4.2.

Листинг 5.4.1 – Вычисление метрик.

```
prfs = precision_recall_fscore_support( y_true=df_test["Class"].
to_numpy(), y_pred=test_pred, labels=dtc.classes_)

print(f"Precision:\t{prfs[0]}")
print(f"Recall:\t{prfs[1]}")
print(f"F1-score:\t{prfs[2]}")
```

Листинг 5.4.2 – Значения метрик.

```
Precision: [0.95454545 0.92307692 0.8125     0.94736842 0.95652174]
Recall:    [0.91304348 0.85714286 0.92857143 0.94736842 0.95652174]
F1-score:  [0.93333333 0.88888889 0.86666667 0.94736842 0.95652174]
```

Из листинга 5.4.2 можно сделать вывод, что значение метрики *Precision* для всех классов принимает значение свыше 90%, наименьшее значение этой метрики наблюдается у класса 2 и равно всего 81%, это говорит о средней вероятности успешного прогнозирования истинного результата. Значение метрики *Recall* также очень большое для всех классов (у большинства классов выше 90%, лишь для класса 1 значение равно 85%), это свидетельствует об очень хорошей вероятности прогнозирования истинного результата на основе истинных значений. Метрика *F1* также имеет очень большое значение, что позволяет судить о хорошей сбалансированности данных и высоком качестве модели.

5.6. Построим полученное дерево решений (см. листинг 5.6.1). Полученное дерево см. на листинге 5.6.1.

Листинг 5.6.1 – Построение дерева решений.

```
1    plot_tree(dtc, feature_names=df.columns[:-1], filled = True)
```

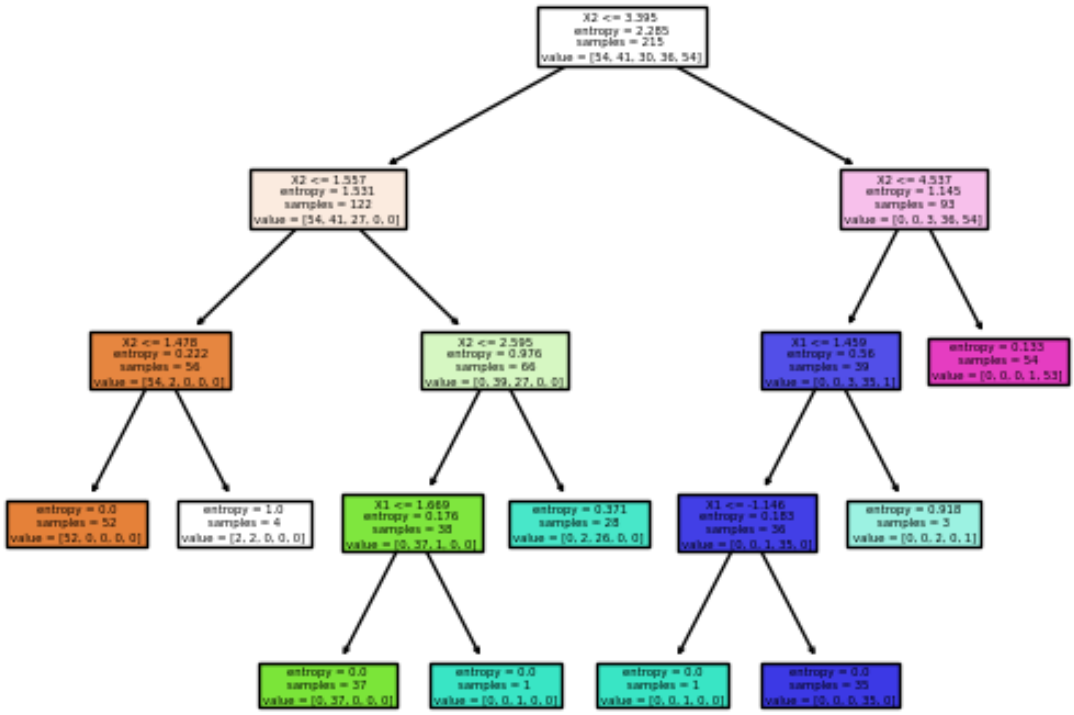

Рисунок 5.6.1 – Дерево решений.

На рисунке 5.6.1 видно процесс построения классификации данных. Каждый узел содержит в себе значение энтропии (загрязнённости), количество точек и веса. Можно видеть, что узлы с малой загрязнённостью имеют яркий цвет, в то время как узлы с высоким значением загрязнённости более тусклые. В дереве есть несколько разветвлений, в каждом из которых находится правило для разделения. Сперва разделение происходит на основании признака “*X2”*, а затем при разделении проверяется значение признака “*X1”*.

6. Выбор классификатора.

6.1. Построим таблицу с метриками *Precision*, *Recall* по каждому классификатору (см. таблицу 6.1.1).

Таблица 6.1.1 – Значения метрик для каждого класса по всем классификаторам.

kNN		
Класс	Precision	Recall
0	0.95652174	0.95652174

1	0.86666667	0.92857143
2	0.92307692	0.85714286
3	1.00000000	0.94736842
4	0.95833333	1.00000000
Логистическая регрессия		
Класс	Precision	Recall
0	0.95454545	0.91304348
1	0.80000000	0.85714286
2	0.92857143	0.92857143
3	1.00000000	0.89473684
4	0.92000000	1.00000000
Метод опорных векторов		
Класс	Precision	Recall
0	0.95833333	1.00000000
1	0.92857143	0.92857143
2	1.00000000	0.92857143
3	0.94736842	0.94736842
4	0.95652174	0.95652174
Решающие деревья		
Класс	Precision	Recall
0	0.95454545	0.91304348
1	0.92307692	0.85714286
2	0.81250000	0.92857143
3	0.94736842	0.94736842
4	0.95652174	0.95652174

Из таблицы 6.1.1 видно, что все классификаторы хорошо справились с задачей классификации. При этом, значение метрики *Precision* стабильно держалось выше 90%, в методах логистической регрессии и решающих деревьев это значение опускается до 80-81%, что также является очень хорошим результатом. Все значения метрики *Recall* также держатся в районе 90-95%, наименьшее зафиксированное значение метрики – 85%, что тоже достаточно хорошо.

6.2. Из полученных результатов можно сделать вывод, что лучше всего на рассматриваемом наборе данных себя показал метод опорных векторов с наибольшими значениями метрик для всех классов. Несмотря на сложную форму, плохую разделяемость и большое количество выбросов у классов исходных

данных, также себя хорошо показали методы kNN и решающих деревьев. С помощью этих методов классы были сбалансированы наиболее правильно.

Выводы.

В ходе выполнения лабораторной работы были изучены различные методы классификации данных.

Проведён анализ и предобработка исходного набора данных.

Для каждого классификатора подобраны параметры, лучше всего подходящие для классификации предоставленного датасета. Построены границы принятия решений, вычислены метрики качества классификации, такие как *Prediction*, *Recall* и *F1*. Для решающего дерева визуализировано и проанализировано дерево решений.

В результате сравнения всех методов было выяснено, что для классификации предоставленного набора данных лучше всего подходит метод опорных векторов, однако остальные методы также хорошо себя показали. Это позволяет заключить, что для задачи классификации исходного набора можно использовать любой из рассмотренных методов, в зависимости от желаемого результата.