**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра БТС**

ИДЗ

**по дисциплине «Технологии и системы принятия решений»**

Тема: Классификация опасных аритмий

Вариант №12

|  |  |  |
| --- | --- | --- |
| Студент гр. 9503 |  | Кошаев Е.А. |
| Преподаватель |  | Манило Л.А. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на ИДЗ**

|  |  |  |
| --- | --- | --- |
| Студент Кошаев Е.А. | | |
| Группа 9503 | | |
| Тема реферата: Классификация опасных аритмий | | |
| Исходные данные:  Отсчёты спектральной плотности мощности коротких НР фрагментов ЭКГ сигнала. Длительность каждого НР фрагмента 2 с, частота дискретизации 360 Гц. Спектр сигнала 0 – 15 Гц, а отсчёты СПМ задаются с шагом 1.4 Гц. Выборка данных включает несколько классов ЭКГ (НР, ЖТ, ФЖ).  Требуется для заданных классов сигналов выполнить следующее:  1) провести классификацию данных методом k ближайших соседей; первые 15 объектов использовать как обучающую выборку, вторые 15 объектов использовать для тестирования;  2) построить решающие правила для распознавания трёх классов объектов, используя методы классификации:  а) случай независимых признаков (по минимуму расстояния);  б) по критерию Фишера (двухклассовая задача);  в) по критерию Фишера (многоклассовая задача).  3) для методов п. 2, а и 2, б определить направление W (ориентирует положение разделяющей гиперплоскости); для метода п. 2, в – плоскость собственных векторов W1, W2.  4) записать уравнения разделяющих гиперплоскостей;  5) отобразить распределение объектов заданных классов в направлении W;  6) вычислив для каждого класса среднее и дисперсию проекций объектов на направление W, получить функции плотности вероятности (использовать нормальный закон распределения);  7) построить ROC кривые; провести сравнение эффективности алгоритмов классификации;  8) записать решающие правила и оценить ошибки классификации (точность распознавания).  Для объектов трех классов свести задачу к поэтапному решению двухклассовых задач. | | |
| Предполагаемый объем реферата:  Не менее 10 страниц. | | |
| Дата выдачи задания: 10.09.2024 | | |
| Дата сдачи реферата: 20.11.2024 | | |
| Дата защиты реферата: 20.11.2024 | | |
|  | | |
| Студент |  | Кошаев Е.А. |
| Преподаватель |  | Манило Л.А. |

**Содержание**

[**1 Подготовка данных 5**](#_Toc182782445)

[**2 Классификация данных методом k ближайших соседей 5**](#_Toc182782446)

[**3 Метод главных компонент 9**](#_Toc182782447)

[**4 Метод классификации по минимуму расстояния 10**](#_Toc182782448)

[**5 Метод классификации по линейному дискриминанту Фишера 13**](#_Toc182782449)

[**6 Метод классификации по дискриминантному анализу с использованием критерия Фишера 17**](#_Toc182782450)

[**7 Сравнение методов 21**](#_Toc182782451)

[**Приложение (основная программа) 24**](#_Toc182782452)

[**Приложение (k ближайших соседей) 32**](#_Toc182782453)

[**Приложение (коэфф. Фишера) 33**](#_Toc182782454)

[**Приложение (дискриминантный анализ) 34**](#_Toc182782455)

# Подготовка данных

Выборка данных включает несколько классов ЭКГ:

* 1 класс - фоновый ритм ФР;
* 2 класс - желудочковая тахикардия ЖТ;
* 3 класс - фибрилляция желудочков ФЖ.

Каждый из трёх классов представлен 30 объектами.

Каждый объект класса характеризуется 11 параметрами. Первый параметр – «Суммарная мощность» – значение полной мощности фрагмента (до 180 Гц) для вычисления в последующем нормированного спектра.

# Классификация данных методом k ближайших соседей

Метод K ближайших соседей основан на гипотезе компактности, которая предполагает, что расположенные близко друг к другу объекты в пространстве признаков имеют схожие значения целевой переменной или принадлежат к одному классу.

В данной работе используется два алгоритма классификации:

1. Объект присваивается тому классу, который является наиболее распространенным среди k соседей данного элемента, классы которых уже известны (т.е. объект имеет наибольшее число соседей в данном классе).
2. Взвешенный способ. Оценивается не только количество объектов, попавших в область близости каждого класса, но и их удаленность от нового объекта. Для каждого класса j определяется оценка близости:

где – расстояние от нового значения до объекта , – количество соседей в данном классе. Объекту присваивается тот класс, у которого выше значение близости.

Чаше всего за метрику расстояния берется евклидово расстояние, которое вычисляется по следующей формуле:

где и – точки, между которыми рассчитывается расстояние.

Для обучения классификаторов используются первые 15 объектов каждого класса. Обучение состоит в запоминании классификатором переданных и размеченных данных. Далее, при классификации объектов, применяется один из описанных выше алгоритмов.

Качество классификации алгоритмов проверялось при помощи подсчета общей точности. Общая точность рассчитывается по формуле:

где – число верно распознанных объектов i-го класса, – число объектов i-го класса, отнесенных к классу j, – число объектов в классе.

На рисунке 1 представлены графики зависимости общей точности от числа ближайших соседей.

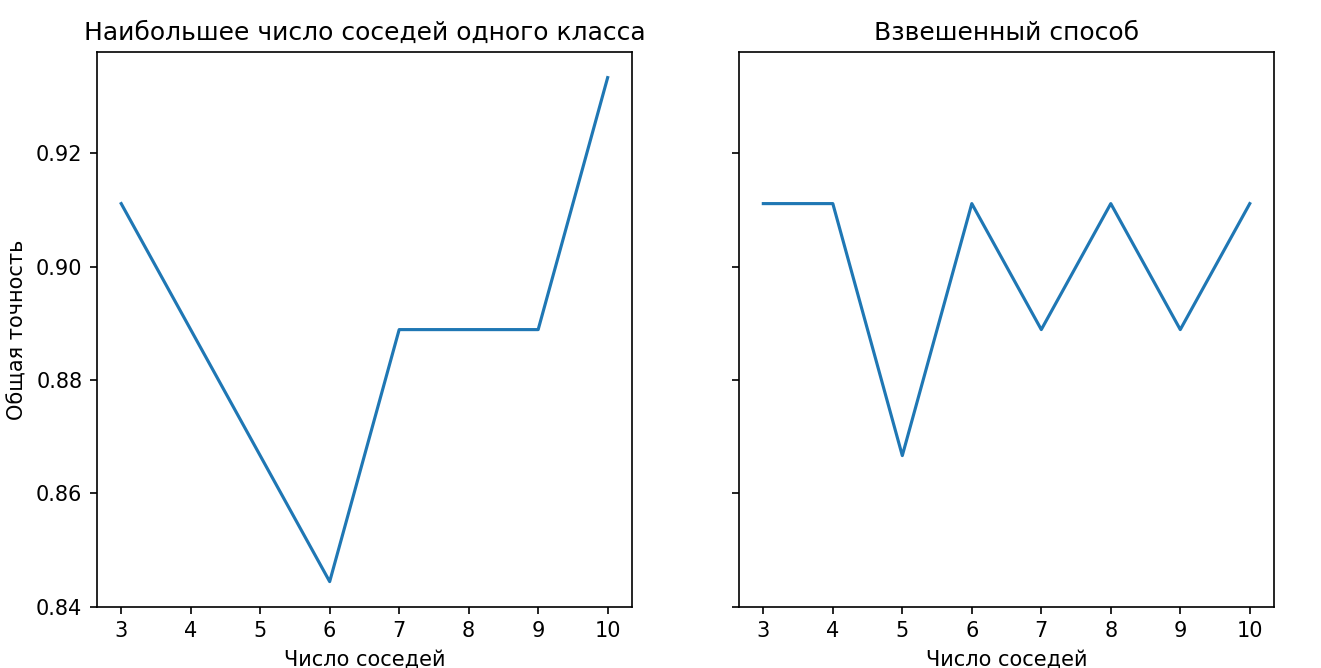


Рисунок 1 – Зависимость общей точности от числа ближайших соседей

В таблицах 1 – 2 представлены результаты классификации с использованием первого и второго алгоритмов соответственно.

Таблица 1 – результаты классификации с использование первого алгоритма.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество соседей,  *K* | Истинный класс,  *i* | Число объектов | Результат распознавания (класс), j | | | Общая точность, OA |
| 1 | 2 | 3 |
| 3 | 1 | 15 | 14 | 1 | 0 | 0.91 |
| 2 | 15 | 3 | 12 | 0 |
| 3 | 15 | 0 | 0 | 15 |
| 4 | 1 | 15 | 14 | 1 | 0 | 0.88 |
| 2 | 15 | 3 | 12 | 0 |
| 3 | 15 | 0 | 1 | 14 |
| 5 | 1 | 15 | 14 | 1 | 0 | 0.86 |
| 2 | 15 | 3 | 10 | 2 |
| 3 | 15 | 0 | 0 | 15 |
| 6 | 1 | 15 | 14 | 1 | 0 | 0.84 |
| 2 | 15 | 4 | 11 | 0 |
| 3 | 15 | 0 | 2 | 13 |
| 7 | 1 | 15 | 14 | 1 | 0 | 0.88 |
| 2 | 15 | 4 | 11 | 0 |
| 3 | 15 | 0 | 0 | 15 |
| 8 | 1 | 15 | 14 | 1 | 0 | 0.88 |
| 2 | 15 | 4 | 11 | 0 |
| 3 | 15 | 0 | 0 | 15 |
| 9 | 1 | 15 | 14 | 1 | 0 | 0.88 |
| 2 | 15 | 3 | 11 | 1 |
| 3 | 15 | 0 | 0 | 15 |
| 10 | 1 | 15 | 15 | 0 | 0 | 0.93 |
| 2 | 15 | 3 | 12 | 0 |
| 3 | 15 | 0 | 0 | 15 |

Таблица 2 – Результаты классификации с использованием второго алгоритма

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество соседей,  *K* | Истинный класс,  *i* | Число объектов | Результат распознавания (класс), j | | | Общая точность, OA |
| 1 | 2 | 3 |
| 3 | 1 | 15 | 14 | 1 | 0 | 0.91 |
| 2 | 15 | 3 | 12 | 0 |
| 3 | 15 | 0 | 0 | 15 |
| 4 | 1 | 15 | 14 | 1 | 0 | 0.91 |
| 2 | 15 | 3 | 12 | 0 |
| 3 | 15 | 0 | 0 | 15 |
| 5 | 1 | 15 | 14 | 1 | 0 | 0.86 |
| 2 | 15 | 3 | 10 | 2 |
| 3 | 15 | 0 | 0 | 15 |
| 6 | 1 | 15 | 14 | 1 | 0 | 0.91 |
| 2 | 15 | 3 | 12 | 0 |
| 3 | 15 | 0 | 0 | 15 |
| 7 | 1 | 15 | 14 | 1 | 0 | 0.88 |
| 2 | 15 | 4 | 11 | 0 |
| 3 | 15 | 0 | 0 | 15 |
| 8 | 1 | 15 | 14 | 1 | 0 | 0.91 |
| 2 | 15 | 3 | 12 | 0 |
| 3 | 15 | 0 | 0 | 15 |
| 9 | 1 | 15 | 14 | 1 | 0 | 0.88 |
| 2 | 15 | 3 | 11 | 1 |
| 3 | 15 | 0 | 0 | 15 |
| 10 | 1 | 15 | 14 | 1 | 0 | 0.91 |
| 2 | 15 | 3 | 12 | 0 |
| 3 | 15 | 0 | 0 | 15 |

# Метод главных компонент

Для дальнейшего исследования методов классификации, сведем трёхклассовую задачу к поэтапному решению двухклассовых задач. Для этого воспользуемся методом главных компонент.

Метод главных компонент позволяет уменьшит пространство признаков, но не потерять их информативность. Это позволит отобразить классы в пространстве двух первых главных компонент, найти наиболее удаленный (или более плотно сгруппированный) класс, отделив его от двух остальных. Оставшиеся два класса объединяются в общий комплексный класс.

Так, на первом этапе решается задача классификации наиболее удаленного класса и комплексного класса, на втором этапе – классификация классов, составляющих общий.

На рисунке 2 представлена диаграмма рассеяния в пространстве двух первых главных компонент.

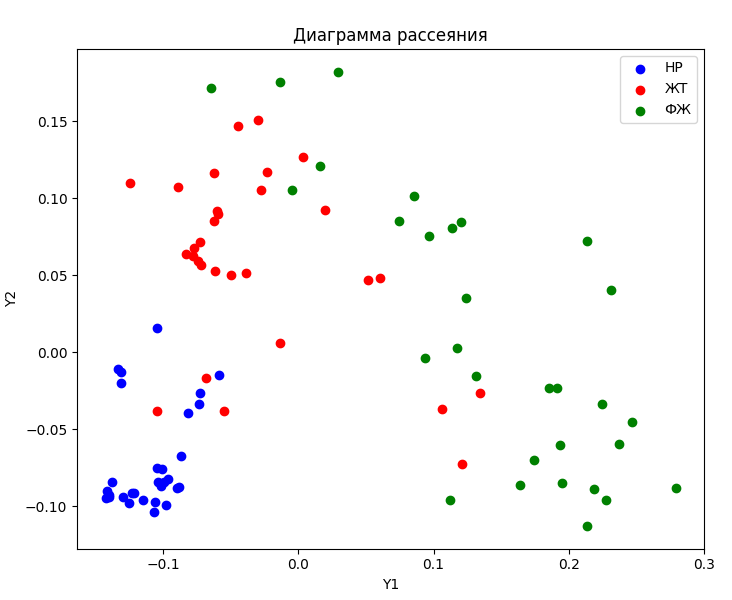


Рисунок 2 – Диаграмма рассеяния в пространстве двух первых главных компонент

Доля дисперсии двух первых главных компонент составляет , . Было решено выделить нормальный ритм (НР) в отдельный класс. Элементы этого класса легко линейно отделимы от двух других классов (фибрилляции желудочков (ФЖ) и желудочковой тахикардии (ЖТ)) как показано на рисунке 2.

Соответственно, на первом этапе будет решаться задача классификации НР и ЖТ + ФЖ, на втором этапе – задача классификации ЖТ и ФЖ.

# Метод классификации по минимуму расстояния

Классификатором по минимуму расстояния называют такой классификатор, оптимальное решающее правило которого формулируется следующим образом: чтобы определить класс вектора *x*, следует измерить евклидово расстояние от *х* до каждого из *c* векторов средних значений и отнести *х* к классу, соответствующему ближайшему среднему значению.

Вычислим весовой вектор из выражения:

где – вектор средних значений по всем признакам для первого класса, – вектор средних значений по всем признакам для второго класса.

Вычисленный весовой вектор необходимо пронормировать.

где – евклидово расстояние. Евклидово расстояние можно рассчитать по формуле .

Процедура распознавания заключается в вычислении проекции вектора на направление весового вектора и сравнении полученной величины с порогом . Порог можно вычислить по следующей формуле:

Так как априорные вероятности по условию равны, то можно представить как . Соответственно:

Однако, более хорошие результаты классификации даёт порог, определенный в месте пресечения функций плотности вероятности. Плотность вероятности можно вычислить следующим образом:

где и – это среднее и стандартное отклонения признака в классе y.

Вычислим для каждого класса среднее и дисперсию проекций объектов на направление W. Результаты вычислений представлены в таблице 3.

Таблица 3 – Результаты вычислений среднего и дисперсии для каждого класса

|  |  |  |  |
| --- | --- | --- | --- |
| Этап классификации | Класс | Среднее, | Дисперсия, |
| Этап 1 (НР и ЖТ+ФЖ) | НР | -0.074 | 0.030 |
| ЖТ+ФЖ | -0.270 | 0.075 |
| Этап 2 (ЖТ и ФЖ) | ЖТ | -0.040 | 0.069 |
| ФЖ | -0.220 | 0.100 |

Разделяющая гиперплоскость определяется, как и ей соответствует следующий алгоритм распознавания:

На рисунке 3 изображены проекции множества классов на весовой вектор для двух этапов.

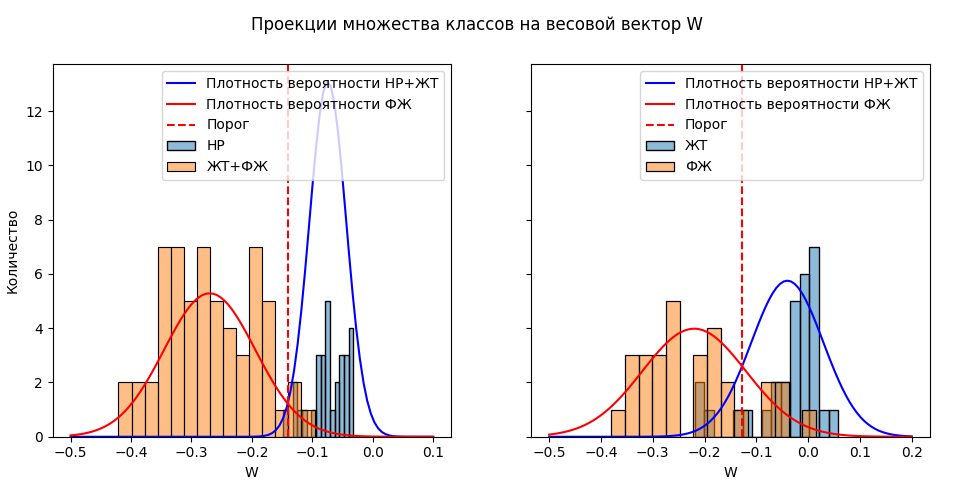


Рисунок 3 – Проекции множества классов на весовой вектор

Проведенные расчеты позволяют получить порог, равный, для первого этапа (НР и ЖТ+ФЖ) , а для второго этапа (ЖТ и ФЖ) .

Запишем уравнения разделяющих гиперплоскостей для первого и второго этапов:

Этап 1 (НР и ЖТ+ФЖ):

Этап 2 (ЖТ и ФЖ):

Решающее правило для каждого из этапов будет иметь следующий вид:

Этап 1 (НР и ЖТ+ФЖ):

Этап 2 (ЖТ и ФЖ):

Можно сформулировать следующий алгоритм классификации: если на первом этапе , то данный объект принадлежит классу НР, иначе объект принадлежит объединённому классу ЖТ+ФЖ и переходит на второй этап классификации. На втором этапе снова сверяем, если , то данный объект принадлежит классу ФЖ, иначе ЖТ.

# Метод классификации по линейному дискриминанту Фишера

Классификатором по линейному дискриминанту Фишера называют такой классификатор, который определяет вектор как линейную функцию с максимальным отношением разброса между классами к «среднему» разбросу внутри классов.

где и – средние значения проекций выборок классов, и – внутриклассовый разброс.

ЛДФ Фишера определяется как такой вектор, для которого критерий максимален.

Среднее значение проекций выборок классов можно представить, как:

где – матрица разброса между классами.

можно рассчитать следующим образом:

Выборочный разброс для двух классов можно определить, как:

где – «усредненная» матрица разброса для двух классов .

Тогда критерий Фишера можно записать в виде:

Искомый вектор можно вычислить, как:

Следует отметить, что полученное выражение вычисления вектора совпадает с выражением для случая нормально распределенных классов с равными ковариационными матрицами, которое записывается следующим образом:

Вычисленный весовой вектор необходимо пронормировать:

Рассчитаем порог, определенный в месте пресечения функций плотности вероятности. Для этого необходимо вычислить для каждого класса среднее и дисперсию проекций объектов на направление W. Результаты вычислений представлены в таблице 4.

Таблица 4 – Результаты вычислений среднего и дисперсии для каждого класса

|  |  |  |  |
| --- | --- | --- | --- |
| Этап классификации | Класс | Среднее, | Дисперсия, |
| Этап 1 (НР и ЖТ+ФЖ) | НР | -0.005 | 0.011 |
| ЖТ+ФЖ | -0.065 | 0.015 |
| Этап 2 (ЖТ и ФЖ) | ЖТ | 0.077 | 0.011 |
| ФЖ | -0.005 | 0.011 |

На рисунке 4 изображены проекции множества классов на весовой вектор для двух этапов.

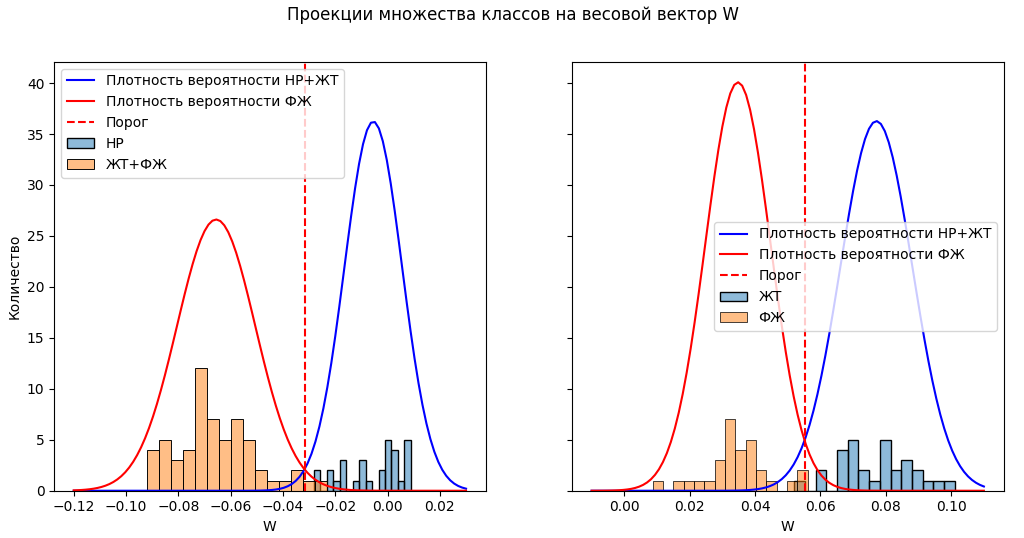


Рисунок 4 – Проекции множества классов на весовой вектор

Проведенные расчеты позволяют получить порог, равный, для первого этапа (НР и ЖТ+ФЖ) , а для второго этапа (ЖТ и ФЖ) .

Запишем уравнения разделяющих гиперплоскостей для первого и второго этапов:

Этап 1 (НР и ЖТ+ФЖ):

Этап 2 (ЖТ и ФЖ):

Решающее правило для каждого из этапов будет иметь следующий вид:

Этап 1 (НР и ЖТ+ФЖ):

Этап 2 (ЖТ и ФЖ):

Можно сформулировать следующий алгоритм классификации: если на первом этапе , то данный объект принадлежит классу НР, иначе объект принадлежит объединённому классу ЖТ+ФЖ и переходит на второй этап классификации. На втором этапе снова сверяем, если, то данный объект принадлежит классу ФЖ, иначе ЖТ.

# Метод классификации по дискриминантному анализу с использованием критерия Фишера

Классификатором по дискриминантному анализу с использованием критерия Фишера называют такой классификатор, который посредствам понижения размерности пространства признаков позволяет производить разделение классов наилучшим образом. Основная идея заключается в предположении о многомерном нормальном распределении признаков внутри классов и поиске их линейного преобразования, которое максимизирует межклассовую дисперсию и минимизирует внутриклассовую.

Данный подход основан на применении того же критерия оптимизации и обобщении определений для матриц разброса между классами и внутри классов. Критерий , оценивающий степень разделения заданных классов сигналов, в общем виде может задаваться следом матрицы

где – матрица рассеяния между классами; – обобщенная матрица рассеяния внутри классов, либо отношением определителей матриц.

Матрица задаётся в следующем виде:

где – частота появления объектов, образующих класс .

Матрица определяется в виде:

где – матрица разброса для i–го класса (находится как выборочная матрица ковариации i-го класса) , – нахождение среднего для i-класса.

Необходимо вычислить матрицы и и далее перейти к нахождению собственных векторов (составляющих матрицы W). Нахождение элементов матрицы сводится к задаче определения собственных значений матрицы . Собственные векторы с ненулевыми собственными значениями определяют то (c-1) c-мерное пространство, в котором могут быть построены дискриминантные функции и определены решающие правила.

Собственные значения вычисленных собственных векторов должны удовлетворять неравенству .

Также, собственные вектора необходимо пронормировать:

На рисунке 5 изображена диаграмма рассеянности классов в уменьшенном пространстве признаков.

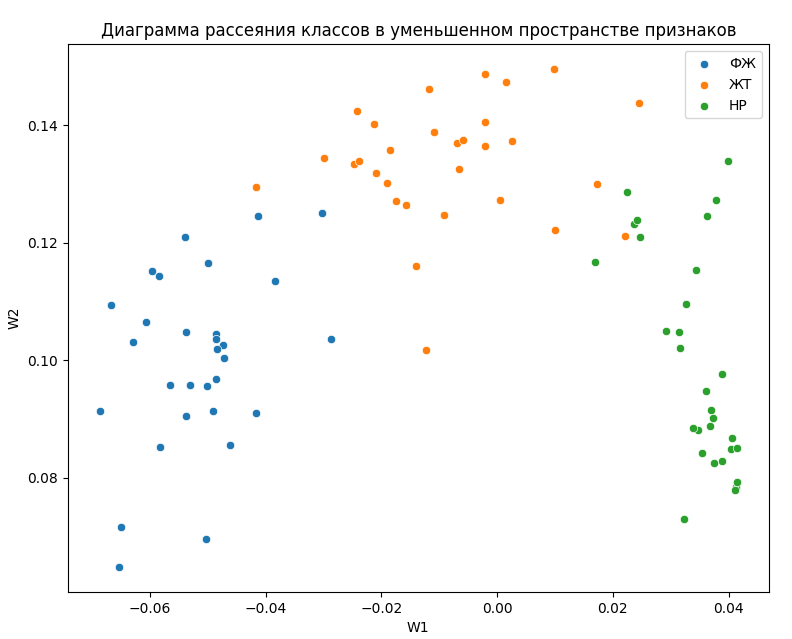


Рисунок 5 – Диаграмма рассеянности классов в уменьшенном пространстве признаков

Решим задачу классификации геометрически. Для этого отделим скопления классов друг от друга при помощи дополнительных геометрических построений.

На рисунке 6 представлена диаграмма рассеянности классов в уменьшенном пространстве признаков с дополнительным построением прямых, разделяющих классы.

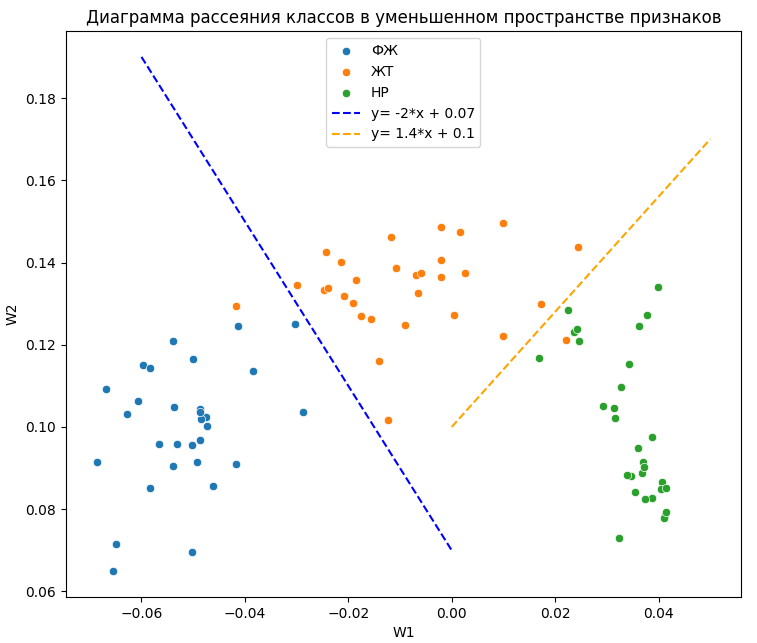


Рисунок 6 – Диаграмма рассеянности классов в уменьшенном пространстве признаков с дополнительным построением прямых, разделяющих классы

Из рисунка 6 видно, что для классификации были построены следующие прямые:

Решающее правило для каждого из этапов будет иметь следующий вид:

Этап 1 (НР и ЖТ+ФЖ):

Этап 2 (ЖТ и ФЖ):

Можно сформулировать следующий алгоритм классификации: если на первом этапе , то данный объект принадлежит классу НР, иначе объект принадлежит объединённому классу ЖТ+ФЖ и переходит на второй этап классификации. На втором этапе снова сверяем, если , то данный объект принадлежит классу ФЖ, иначе ЖТ.

# Сравнение методов

Сравним методы классификации, используемые в данной работе. Для этого построим ROC-кривые для каждого из этапов классификации. Кривая ROC – это график, который иллюстрирует производительность классификационной модели при всех возможных порогах классификации. Ось X данного графика представляет собой специфичность, т.е ложноположительную частоту, а ось Y – чувствительность, т.е истинно положительную частоту.

Чувствительность и специфичность можно рассчитать следующим образом:

где – истинно положительные результаты, – ложно отрицательные результаты, – ложноположительные результаты, – истинно отрицательные результаты.

На рисунке 7 изображены ROC-кривые каждого классификатора для первого и второго этапов.

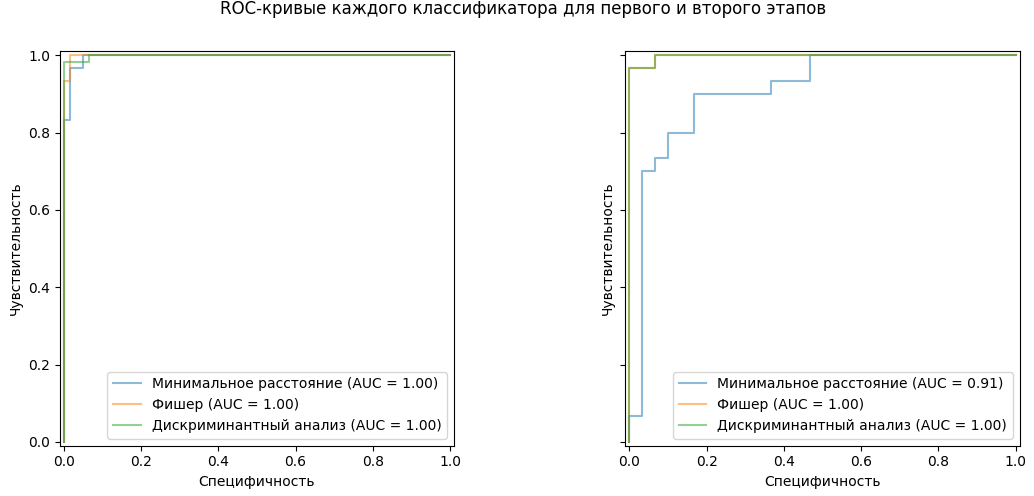


Рисунок 7 – ROC-кривые каждого классификатора для первого и второго этапов

Анализируя результаты, приведенные на рисунке 7, можно заменить, что дискриминантный анализ показывает наилучшие результаты для обоих этапов классификации. Также, следует заметить, что с первым этапом классификации (НР и ЖТ+ФЖ) отлично справились все 3 классификатора, а на втором хорошие результаты также показывает классификатор по Фишеру.

В таблицах 5-7 приведены расчётные значения метрик для каждого классификатора.

Таблица 5 – Метрики классификатора по минимуму расстояния

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Этап классификации | Класс | TP | TN | FN | FP | Чувствительность,  Se | Специфичность,  Sp | Точность,  Acc |
| Этап 1 (НР и ЖТ+ФЖ) | НР | 29 | 57 | 1 | 3 | 96% | 95% | 95% |
| ЖТ+ФЖ | 57 | 29 | 3 | 1 | 95% | 96% |
| Этап 2 (ЖТ и ФЖ) | ЖТ | 26 | 25 | 4 | 5 | 86% | 83% | 85% |
| ФЖ | 25 | 26 | 5 | 4 | 83% | 86% |

Таблица 6 – Метрики классификатора по критерию Фишера

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Этап классификации | Класс | TP | TN | FN | FP | Чувствительность,  Se | Специфичность,  Sp | Точность, |
| Этап 1 (НР и ЖТ+ФЖ) | НР | 30 | 58 | 0 | 2 | 100% | 96% | 97% |
| ЖТ+ФЖ | 58 | 30 | 2 | 0 | 96% | 100% |
| Этап 2 (ЖТ и ФЖ) | ЖТ | 29 | 29 | 1 | 1 | 96% | 96% | 96% |
| ФЖ | 29 | 29 | 1 | 1 | 96% | 96% |

Таблица 7 – Метрики классификатора по дискриминантному анализу

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Этап классификации | Класс | TP | TN | FN | FP | Чувствительность,  Se | Специфичность,  Sp | Точность, |
| Этап 1 (НР и ЖТ+ФЖ) | НР | 30 | 59 | 0 | 1 | 100% | 98% | 98% |
| ЖТ+ФЖ | 59 | 30 | 1 | 0 | 98% | 100% |
| Этап 2 (ЖТ и ФЖ) | ЖТ | 29 | 30 | 1 | 0 | 96% | 100% | 98% |
| ФЖ | 30 | 29 | 0 | 1 | 100% | 96% |

Анализирую данные в таблицах 5-7, можно увидеть, что классификатор по дискриминантному анализу показывает наилучшие результаты по таким параметрам, как: точность, чувствительность, специфичность. Что подтверждает выводе, сделанный ранее.

Также, следует проанализировать результаты, полученные при использовании метода ближайших соседей. Анализируя таблицы 1-2 и рисунок 1 можно сказать, что наибольшую точность классификации показала первая реализация метода k ближайших соседей. Однако, его график зависимости точности от числа соседей весьма хаотичен. При использовании взвешенного метода k ближайших соседей наибольшая точность достигнута при k = 7. Можно заметить, что при разных значения k взвешенный вариант дает более стабильные показатели точности, впрочем, максимальная точность оказалась ниже.

Однако, точность классификации, полученная данным методом ниже точности классификации, полученной при использовании классификатора по дискриминантному анализу. Соответственно, можно сделать вывод, что классификатор по дискриминантному анализу показывает наилучшие результаты классификации в сравнении с другими методами, рассмотренными в данной работе.

# Приложение (основная программа)

from utils.k\_nearest\_neighbours import KNearestNeighbors

from utils.fisher import Fisher

from utils.minimal\_distance import MinDistance

from utils.prepare\_data import prepare\_data

from utils.calc\_params import calc\_params, calc\_stats

from utils.mda import MDA

from utils.plot\_funcs import plot\_hist

from sklearn.decomposition.\_pca import PCA

from sklearn.metrics import accuracy\_score, RocCurveDisplay

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

data\_filepath = './data/data.xlsx'

x\_total, y\_total = prepare\_data(data\_filepath)

NR = x\_total[:30]

JTFJ = x\_total[30:]

FJ = JTFJ[30:]

JT = JTFJ[:30]

NRJTFJ\_train = pd.concat([NR[:15], JT[:15], FJ[:15]], ignore\_index=True)

y = pd.concat([y\_total[:15], y\_total[30:45], y\_total[60:75]], ignore\_index=True)

NRJTFJ\_test = pd.concat([NR[15:], JT[15:], FJ[15:]], ignore\_index=True)

KNN = KNearestNeighbors()

KNN.fit(NRJTFJ\_train, y)

KNN\_p = KNearestNeighbors(mode="proximity")

KNN\_p.fit(NRJTFJ\_train, y)

KNN\_res = []

KNN\_p\_res = []

for i in range(3, 11):

res = KNN.predict(i, NRJTFJ\_test)

res\_p = KNN\_p.predict(i, NRJTFJ\_test)

KNN\_res.append(accuracy\_score(y, res))

KNN\_p\_res.append(accuracy\_score(y, res\_p))

# print(f'K = {i}, KNN, OA = {accuracy\_score(y, res)}')

# print(confusion\_matrix(y, res))

# print(f'K = {i}, KNN\_P, OA = {accuracy\_score(y, res\_p)}')

# print(confusion\_matrix(y, res\_p))

fig, axes = plt.subplots(1, 2, sharey=True)

axes[0].plot(np.linspace(3,10, num=8), KNN\_res)

axes[0].set\_title('Наибольшее число соседей одного класса')

axes[0].set\_ylabel("Общая точность")

axes[0].set\_xlabel("Число соседей")

axes[1].plot(np.linspace(3,10, num=8), KNN\_p\_res)

axes[1].set\_title('Взвешенный способ')

axes[1].set\_xlabel("Число соседей")

plt.show()

pca = PCA(n\_components=2)

data\_reduced\_space = pca.fit\_transform(x\_total.values)

print(f"ГК = {pca.explained\_variance\_ratio\_}")

# Display PCA

fig, ax = plt.subplots()

ax.scatter(data\_reduced\_space[:30, 0], data\_reduced\_space[:30, 1], c=['blue'], label="НР")

ax.scatter(data\_reduced\_space[30:60, 0], data\_reduced\_space[30:60, 1], c=['red'], label="ЖТ")

ax.scatter(data\_reduced\_space[60:90, 0], data\_reduced\_space[60:90, 1], c=['green'], label="ФЖ")

ax.set\_xlabel("Y1")

ax.set\_ylabel("Y2")

ax.set\_title("Диаграмма рассеяния")

ax.legend()

plt.show()

# MinDistance

md\_NR = MinDistance(0, 1)

md\_NR.fit(NR.values, JTFJ.values)

pred\_NR = md\_NR.predict(NR.values)

pred\_JTFJ = md\_NR.predict(JTFJ.values)

NR\_mean, NR\_std = calc\_params(pred\_NR)

JTFJ\_mean, JTFJ\_std = calc\_params(pred\_JTFJ)

md\_1\_pred = np.append(pred\_NR, pred\_JTFJ)

\_md\_1\_binary\_pred = np.array([1 if x > -0.14 else 0 for x in md\_1\_pred])

TPR, FPR, ACC, FP, FN, TP, TN = calc\_stats(np.concatenate((np.ones(30, dtype=int), np.zeros(60, dtype=int))), \_md\_1\_binary\_pred)

# print("~~~~ Nr/JTFJ ~~~~")

# print(f"{NR\_mean=} {NR\_std=}")

# print(f"{JTFJ\_mean=} {JTFJ\_std=}")

md\_JTFJ = MinDistance(0, 1)

md\_JTFJ.fit(JT.values, FJ.values)

pred\_JT = md\_JTFJ.predict(JT.values)

pred\_FJ = md\_JTFJ.predict(FJ.values)

JT\_mean, JT\_std = calc\_params(pred\_JT)

FJ\_mean, FJ\_std = calc\_params(pred\_FJ)

md\_2\_pred = np.append(pred\_JT, pred\_FJ)

# print("~~~~ JT/FJ ~~~~")

# print(f"{JT\_mean=} {JT\_std=}")

# print(f"{FJ\_mean=} {FJ\_std=}")

fig, axes = plt.subplots(1, 2, sharey=True)

x\_values = np.linspace(-0.5, 0.1, num=100)

\_pred\_NR = { "x": pred\_NR, "hue": "НР" }

\_pred\_JTFJ = { "x": pred\_JTFJ, "hue": "ЖТ+ФЖ" }

fisher = Fisher()

fisher.W = md\_NR.vector\_w

threshold = fisher.\_find\_threshold(NR.values, JTFJ.values)

plot\_hist(x\_values, axes[0], -0.14, \_pred\_NR, \_pred\_JTFJ)

# print(f"{md\_NR.vector\_w=}")

# print(f"{-0.14}")

# \_md\_1\_binary\_pred = np.array([1 if x > -0.14 else 0 for x in md\_1\_pred])

# TPR, FPR, ACC, FP, FN, TP, TN = calc\_stats(np.concatenate((np.ones(30, dtype=int), np.zeros(60, dtype=int))), \_md\_1\_binary\_pred)

# print(f"{TPR=}")

# print(f"{FPR=}")

# print(f"{ACC=}")

# print(f"{FP=}")

# print(f"{FN=}")

# print(f"{TP=}")

# print(f"{TN=}")

x\_values = np.linspace(-0.5, 0.2, num=100)

\_pred\_JT = { "x": pred\_JT, "hue": "ЖТ" }

\_pred\_FJ = { "x": pred\_FJ, "hue": "ФЖ" }

fisher = Fisher()

fisher.W = md\_JTFJ.vector\_w

threshold = fisher.\_find\_threshold(JT.values, FJ.values)

plot\_hist(x\_values, axes[1], threshold, \_pred\_JT, \_pred\_FJ)

# print(f"{md\_JTFJ.vector\_w=}")

# print(f"{threshold=}")

# \_md\_2\_binary\_pred = np.array([1 if x > threshold else 0 for x in md\_2\_pred])

# TPR, FPR, ACC, FP, FN, TP, TN = calc\_stats(np.concatenate((np.ones(30, dtype=int), np.zeros(30, dtype=int))), \_md\_2\_binary\_pred)

# print(f"{TPR=}")

# print(f"{FPR=}")

# print(f"{ACC=}")

# print(f"{FP=}")

# print(f"{FN=}")

# print(f"{TP=}")

# print(f"{TN=}")

# \_NRJT = np.append(pred\_NR, pred\_JT)

# md\_2\_pred = np.array([1 if x < threshold else 0 for x in \_NRJT])

fig.suptitle("Проекции множества классов на весовой вектор W")

plt.show()

# Fisher

fisher\_NR = Fisher()

fisher\_NR.fit(NR.values, JTFJ.values)

pred\_JTFJ = fisher\_NR.predict(JTFJ.values)

pred\_NR = fisher\_NR.predict(NR.values)

JTFJ\_mean, JTFJ\_std = calc\_params(pred\_JTFJ)

NR\_mean, NR\_std = calc\_params(pred\_NR)

fish\_1\_pred = np.append(pred\_NR, pred\_JTFJ)

# print("~~~~ NR/JTFJ ~~~~")

# print(f"{NR\_mean=} {NR\_std=}")

# print(f"{JTFJ\_mean=} {JTFJ\_std=}")

# print(f"{fisher\_NR.threshold}")

# print(f"{fisher\_NR.W=}")

# \_fish\_1\_binary\_pred = np.array([1 if x > fisher\_NR.threshold else 0 for x in fish\_1\_pred])

# TPR, FPR, ACC, FP, FN, TP, TN = calc\_stats(np.concatenate((np.ones(30, dtype=int), np.zeros(60, dtype=int))), \_fish\_1\_binary\_pred)

# print(f"{TPR=}")

# print(f"{FPR=}")

# print(f"{ACC=}")

# print(f"{FP=}")

# print(f"{FN=}")

# print(f"{TP=}")

# print(f"{TN=}")

fisher\_JTFJ = Fisher()

fisher\_JTFJ.fit(JT.values, FJ.values)

pred\_FJ = fisher\_JTFJ.predict(FJ.values)

pred\_JT = fisher\_JTFJ.predict(JT.values)

FJ\_mean, FJ\_std = calc\_params(pred\_NR)

JT\_mean, JT\_std = calc\_params(pred\_JT)

fish\_2\_pred = np.append(pred\_JT, pred\_FJ)

# print("~~~~ JT/FJ ~~~~")

# print(f"{FJ\_mean=} {FJ\_std=}")

# print(f"{JT\_mean=} {JT\_std=}")

# print(f"{fisher\_JTFJ.threshold=}")

# print(f"{fisher\_JTFJ.W=}")

# \_fish\_2\_binary\_pred = np.array([1 if x > fisher\_JTFJ.threshold else 0 for x in fish\_2\_pred])

# TPR, FPR, ACC, FP, FN, TP, TN = calc\_stats(np.concatenate((np.ones(30, dtype=int), np.zeros(30, dtype=int))), \_fish\_2\_binary\_pred)

# print(f"{TPR=}")

# print(f"{FPR=}")

# print(f"{ACC=}")

# print(f"{FP=}")

# print(f"{FN=}")

# print(f"{TP=}")

# print(f"{TN=}")

fig, axes = plt.subplots(1, 2, sharey=True)

x\_values = np.linspace(-0.12, 0.03, num=100)

\_pred\_NR = { "x": pred\_NR, "hue": "НР" }

\_pred\_JTFJ = { "x": pred\_JTFJ, "hue": "ЖТ+ФЖ" }

plot\_hist(x\_values, axes[0], fisher\_NR.threshold, \_pred\_NR, \_pred\_JTFJ)

# fish\_1\_pred = np.append(pred\_FJ, pred\_NRJT)

# fish\_1\_fpr, fish\_1\_tpr, t = roc\_curve(np.concatenate((np.ones(30, dtype=int), np.zeros(60, dtype=int))), fish\_1\_pred)

# fish\_1\_pred = np.array([1 if x < fisher\_FJ.threshold else 0 for x in \_FJNRjt])

x\_values = np.linspace(-0.01, 0.11, num=100)

\_pred\_JT = { "x": pred\_JT, "hue": "ЖТ" }

\_pred\_FJ = { "x": pred\_FJ, "hue": "ФЖ" }

plot\_hist(x\_values, axes[1], fisher\_JTFJ.threshold, \_pred\_JT, \_pred\_FJ)

# \_FJNRjt = np.append(pred\_NR, pred\_JT)

# fish\_2\_pred = np.array([1 if x < fisher\_NRJT.threshold else 0 for x in \_FJNRjt])

fig.suptitle("Проекции множества классов на весовой вектор W")

plt.show()

# MDA

mda = MDA()

mda.fit(NR.values, JT.values, FJ.values)

x\_FJ, y\_FJ = mda.predict(FJ.values)

x\_JT, y\_JT = mda.predict(JT.values)

x\_NR, y\_NR = mda.predict(NR.values)

df = { 'W1': x\_FJ, 'W2': y\_FJ }

sns.scatterplot(data=df, x='W1', y='W2', label="ФЖ")

df = { 'W1': x\_JT, 'W2': y\_JT }

sns.scatterplot(data=df, x='W1', y='W2', label="ЖТ")

df = { 'W1': x\_NR, 'W2': y\_NR }

sns.scatterplot(data=df, x='W1', y='W2', label="НР")

x\_values\_1 = np.linspace(-0.06, 0, num=100)

y\_values\_1 = -2\*x\_values\_1 + 0.07

x\_values\_2 = np.linspace(0, 0.05, num=100)

y\_values\_2 = 1.4\*x\_values\_2 + 0.1

sns.lineplot(x=x\_values\_1, y=y\_values\_1, label="y= -2\*x + 0.07", c="blue", linestyle='--')

sns.lineplot(x=x\_values\_2, y=y\_values\_2, label="y= 1.4\*x + 0.1", c="orange", linestyle='--')

plt.title("Диаграмма рассеяния классов в уменьшенном пространстве признаков")

plt.legend()

plt.show()

\_NR = np.hstack((x\_NR.reshape(len(x\_NR), 1), y\_NR.reshape(len(y\_NR), 1)))

\_JTFJ = np.hstack((np.append(x\_JT, x\_FJ).reshape(60, 1), np.append(y\_JT, y\_FJ).reshape(60, 1)))

\_FJNRjt = np.vstack((\_NR, \_JTFJ))

res\_FJNRjt = np.array([t[1] - 1.4\*t[0] - 0.1 for t in \_FJNRjt])

# \_res\_1\_binary\_pred = np.array([1 if t[1] - 1.4\*t[0] - 0.1 < 0 else 0 for t in \_FJNRjt])

# TPR, FPR, ACC, FP, FN, TP, TN = calc\_stats(np.concatenate((np.ones(30, dtype=int), np.zeros(60, dtype=int))), \_res\_1\_binary\_pred)

# print(f"{TPR=}")

# print(f"{FPR=}")

# print(f"{ACC=}")

# print(f"{FP=}")

# print(f"{FN=}")

# print(f"{TP=}")

# print(f"{TN=}")

\_FJ = np.hstack((x\_FJ.reshape(len(x\_FJ), 1), y\_FJ.reshape(len(y\_FJ), 1)))

\_JT = np.hstack((x\_JT.reshape(len(x\_JT), 1), y\_JT.reshape(len(y\_JT), 1)))

\_JTFJ = np.vstack((\_JT, \_FJ))

res\_NRJT = np.array([t[1] + 2\*t[0] - 0.07 for t in \_JTFJ])

# \_res\_2\_binary\_pred = np.array([1 if t[1] + 2\*t[0] - 0.07 > 0 else 0 for t in \_JTFJ])

# TPR, FPR, ACC, FP, FN, TP, TN = calc\_stats(np.concatenate((np.ones(30, dtype=int), np.zeros(30, dtype=int))), \_res\_2\_binary\_pred)

# print(f"{TPR=}")

# print(f"{FPR=}")

# print(f"{ACC=}")

# print(f"{FP=}")

# print(f"{FN=}")

# print(f"{TP=}")

# print(f"{TN=}")

fig, axes = plt.subplots(1, 2, sharey=True)

RocCurveDisplay.from\_predictions(np.concatenate((np.ones(30, dtype=int), np.zeros(60, dtype=int))), md\_1\_pred, ax=axes[0], alpha=0.5, name="Минимальное расстояние")

RocCurveDisplay.from\_predictions(np.concatenate((np.ones(30, dtype=int), np.zeros(60, dtype=int))), fish\_1\_pred, ax=axes[0], alpha=0.5, name="Фишер")

RocCurveDisplay.from\_predictions(np.concatenate((np.zeros(30, dtype=int), np.ones(60, dtype=int))), res\_FJNRjt, ax=axes[0], alpha=0.5, name="Дискриминантный анализ")

RocCurveDisplay.from\_predictions(np.concatenate((np.ones(30, dtype=int), np.zeros(30, dtype=int))), md\_2\_pred, ax=axes[1], alpha=0.5, name="Минимальное расстояние")

RocCurveDisplay.from\_predictions(np.concatenate((np.ones(30, dtype=int), np.zeros(30, dtype=int))), fish\_2\_pred, ax=axes[1], alpha=0.5, name="Фишер")

RocCurveDisplay.from\_predictions(np.concatenate((np.ones(30, dtype=int), np.zeros(30, dtype=int))), res\_NRJT, ax=axes[1], alpha=0.5, name="Дискриминантный анализ")

axes[0].set\_ylabel("Чувствительность")

axes[1].set\_ylabel("Чувствительность")

axes[0].set\_xlabel("Специфичность")

axes[1].set\_xlabel("Специфичность")

fig.suptitle("ROC-кривые каждого классификатора для первого и второго этапов")

plt.show()

# Приложение (k ближайших соседей)

import numpy as np

import pandas as pd

class KNearestNeighbors:

def \_\_init\_\_(self, mode="distance") -> None:

self.mode = mode

def fit(self, x\_train: pd.DataFrame, y\_train: pd.DataFrame) -> None:

self.x\_train = x\_train

self.y\_train = y\_train

def \_euclidean\_distances(self, X, x\_test\_i) -> None:

return np.sqrt(np.sum((X - x\_test\_i) \*\* 2, axis=1))

def \_make\_prediction\_distance(self, x\_test\_i):

distance = self.\_euclidean\_distances(self.x\_train, x\_test\_i)

k\_nearest\_indexes = np.argsort(distance)[:self.n\_neighbors]

targets = self.y\_train.iloc[[\*k\_nearest\_indexes]]

return np.bincount(targets.values.ravel()).argmax()

def \_make\_prediction\_proximity(self, x\_test\_i):

distance = self.\_euclidean\_distances(self.x\_train, x\_test\_i)

k\_nearest\_indexes = np.argsort(distance)[:self.n\_neighbors]

# Filter the nearest neighbors based on class

nearest\_neighbors = self.x\_train.iloc[k\_nearest\_indexes]

nearest\_labels = self.y\_train.iloc[k\_nearest\_indexes, 0]

# Split the DataFrame by class

cls\_1 = nearest\_neighbors[nearest\_labels == 0]

cls\_2 = nearest\_neighbors[nearest\_labels == 1]

cls\_3 = nearest\_neighbors[nearest\_labels == 2]

proximity\_1 = self.\_calc\_proximity(pd.DataFrame(cls\_1), x\_test\_i)

proximity\_2 = self.\_calc\_proximity(pd.DataFrame(cls\_2), x\_test\_i)

proximity\_3 = self.\_calc\_proximity(pd.DataFrame(cls\_3), x\_test\_i)

max\_proximity = max(proximity\_1, proximity\_2, proximity\_3)

if max\_proximity == proximity\_1:

return 0

elif max\_proximity == proximity\_2:

return 1

else:

return 2

def \_calc\_proximity(self, cls, x\_test\_i):

return np.sum(1/np.sqrt(self.\_euclidean\_distances(cls, x\_test\_i)))

def predict(self, n\_neighbors: int, X\_test: pd.DataFrame):

self.n\_neighbors = n\_neighbors

if (self.mode == "distance"):

return np.array([self.\_make\_prediction\_distance(row) for index, row in X\_test.iterrows()])

elif (self.mode == "proximity"):

return np.array([self.\_make\_prediction\_proximity(row) for index, row in X\_test.iterrows()])

# Приложение (коэфф. Фишера)

import numpy as np

class Fisher():

def \_\_init\_\_(self) -> None:

self.W = None

def fit(self, class\_1:np.ndarray, class\_2:np.ndarray):

diff\_mean = np.mean(class\_1, axis=0) - np.mean(class\_2, axis=0)

sum\_covariance = np.cov(class\_1, rowvar=0) + np.cov(class\_2, rowvar=0)

if sum\_covariance.ndim < 1:

sum\_covariance = np.array([[sum\_covariance]])

self.W = np.matmul(np.linalg.inv(sum\_covariance), diff\_mean)

self.W = self.W/np.linalg.norm(self.W)

self.threshold = self.\_find\_threshold(class\_1, class\_2)

return self

def predict(self, data:np.ndarray):

proj = np.matmul(data, self.W)

return proj

# return np.where(proj > self.threshold, 1, 0)

def \_find\_threshold(self, class\_1:np.ndarray, class\_2:np.ndarray):

proj1 = np.matmul(class\_1, self.W)

proj2 = np.matmul(class\_2, self.W)

proj1\_mean = np.mean(proj1)

proj2\_mean = np.mean(proj2)

proj1\_std = np.std(proj1)

proj2\_std = np.std(proj2)

threshold = self.\_solve\_pdfs(proj1\_mean, proj2\_mean, proj1\_std, proj2\_std)

return threshold

def \_solve\_pdfs(self, pdf\_mean\_1, pdf\_mean\_2, pdf\_std\_1, pdf\_std\_2):

coeff1 = 1/(2\*pdf\_std\_1\*\*2) - 1/(2\*pdf\_std\_2\*\*2)

coeff2 = pdf\_mean\_2/(pdf\_std\_2\*\*2) - pdf\_mean\_1/(pdf\_std\_1\*\*2)

coeff3 = pdf\_mean\_1\*\*2 /(2\*pdf\_std\_1\*\*2) - pdf\_mean\_2\*\*2 / (2\*pdf\_std\_2\*\*2) - np.log(pdf\_std\_2/pdf\_std\_1)

coeffs = [coeff1, coeff2, coeff3]

roots\_of\_eq = np.roots(coeffs)

threshold = roots\_of\_eq[1]

return threshold

# Приложение (дискриминантный анализ)

import numpy as np

class MDA:

def \_\_init\_\_(self) -> None:

return None

def fit(self, class\_1: np.ndarray, class\_2: np.ndarray, class\_3: np.ndarray):

cov\_1 = np.cov(class\_1, rowvar=0)

M\_1 = np.mean(class\_1, axis=0)

cov\_2 = np.cov(class\_2, rowvar=0)

M\_2 = np.mean(class\_2, axis=0)

cov\_3 = np.cov(class\_3, rowvar=0)

M\_3 = np.mean(class\_3, axis=0)

M = np.mean([\*class\_1, \*class\_2, \*class\_3], axis=0)

Sw = (1/3)\*(cov\_1 + cov\_2 + cov\_3)

Sb = 0

for i in [M\_1, M\_2, M\_3]:

Sb = Sb + (1/3)\*np.matmul((i - M).reshape(len(class\_1[0]), 1),(i - M).reshape((1, len(class\_1[0]))))

W = np.linalg.inv(Sw).dot(Sb)

eigenvalues, eigenvectors = np.linalg.eig(W)

idx = np.argsort(eigenvalues)[::-1] # Sort in descending order

eigenvectors = eigenvectors[:, idx] # Sort eigenvectors

self.W\_1 = eigenvectors[:, 0] / np.linalg.norm(eigenvectors[:, 0])

self.W\_2 = eigenvectors[:, 1] / np.linalg.norm(eigenvectors[:, 1])

def predict(self, cs: np.ndarray):

pred\_w1 = np.matmul(self.W\_1, cs.T)

pred\_w2 = np.matmul(self.W\_2, cs.T)

return pred\_w1, pred\_w2