МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Допущен к защите

Заведующий кафедрой ПМИ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Разова

**Исследование метрических методов классификации**

Курсовой проект по дисциплине  
«Проектная и научно-исследовательская деятельность»

Выполнил студент группы ФИб-2301-51-00     / Д. О. Ощепков/

Руководитель к.п.н., профессор кафедры ПМИ     / Е. В. Котельников/

Работа защищена с оценкой     \_\_\_\_.\_\_\_\_.2023 г.

Члены комиссии:     / /

    / /

Киров 2023

Содержание

[Введение 4](#_Toc137232959)

[1 Разбор метрических методов классификации 7](#_Toc137232960)

[1.1 Постановка задачи 7](#_Toc137232961)

[1.2 Метрики оценки качества 7](#_Toc137232962)

[1.2.1 Precision и recall 8](#_Toc137232963)

[1.2.2 метрика и ее частный случай метрика 9](#_Toc137232964)

[1.2.3 метрика и ее частный случай метрика 9](#_Toc137232965)

[1.3 Вычислительная сложность рассматриваемых алгоритмов 9](#_Toc137232966)

[1.4 Метод ближайшего соседа 11](#_Toc137232967)

[1.4.1 Формулировка 11](#_Toc137232968)

[1.4.2 Достоинства и недостатки 11](#_Toc137232969)

[1.4.3 Алгоритм 12](#_Toc137232970)

[1.5 Метод k ближайших соседей 12](#_Toc137232971)

[1.5.1 Формулировка 12](#_Toc137232972)

[1.5.2 Алгоритм 13](#_Toc137232973)

[1.5.3 Достоинства и недостатки 14](#_Toc137232974)

[1.6 Метод взвешенных k ближайших соседей 15](#_Toc137232975)

[1.6.1 Формулировка 15](#_Toc137232976)

[1.6.2 Алгоритм 16](#_Toc137232977)

[1.6.3 Достоинства и недостатки 17](#_Toc137232978)

[1.7 Метод парзеновского окна фиксированной ширины 17](#_Toc137232979)

[1.7.1 Формулировка 17](#_Toc137232980)

[1.7.2 Алгоритм 18](#_Toc137232981)

[1.7.3 Преимущества и недостатки 19](#_Toc137232982)

[1.8 Метод парзеновского окна переменной ширины ширины 19](#_Toc137232983)

[1.8.1 Формулировка 19](#_Toc137232984)

[1.8.2 Алгоритм 21](#_Toc137232985)

[1.8.3 Преимущества и недостатки 22](#_Toc137232986)

[1.9 Метод потенциальных функций 22](#_Toc137232987)

[1.9.1 Формулировка 22](#_Toc137232988)

[1.9.2 Алгоритм 23](#_Toc137232989)

[1.9.3 Преимущества и недостатки 24](#_Toc137232990)

[1.10 Выводы по разделу 25](#_Toc137232991)

[2 Применение алгоритмов на практике 27](#_Toc137232992)

[2.1 Bank Customer Churn Dataset 27](#_Toc137232993)

[2.1.1 Обзор и анализ датасета 27](#_Toc137232994)

[2.2 Diabetes prediction dataset 31](#_Toc137232995)

[2.3 Mobile Price Classification 37](#_Toc137232996)

[2.4 Выводы по разделу 42](#_Toc137232997)

[Заключение 43](#_Toc137232998)

[Библиографический список 44](#_Toc137232999)

[Приложения 45](#_Toc137233000)

[Приложение А. Листинг программы 45](#_Toc137233001)

[Приложение Б. Перечень сокращений и обозначений 46](#_Toc137233002)

# Введение

Метрические методы классификации относятся к области обучения с учителем и основываются на измерении расстояния или сходства между объектами. Задача состоит в разделении данных на заранее определенные классы на основе известных обучающих примеров.

Метрические методы классификации применяются в различных областях жизнедеятельности человека:

* Биомедицина и медицинская диагностика. Метрические методы классификации находят применение в задачах биомедицинской классификации и диагностики, например, в определении заболеваний по медицинским данным или анализе медицинских изображений [1].
* Обработка естественного языка (Natural Language Processing, NLP). В задачах анализа текста и обработки естественного языка метрические методы классификации используются для распознавания и классификации текстовых данных, например, в определении тональности текста или автоматической классификации текстовых документов [2].
* Компьютерное зрение и обработка изображений. В области компьютерного зрения метрические методы классификации используются для распознавания и классификации изображений, например, в задачах распознавания образов или детектирования объектов на изображениях [3].

Основные подходы в метрических методах классификации включают:

1. Метод одного ближайшего соседа (1-Nearest Neighbor). В этом методе объект классифицируется на основе меток его ближайшего соседей в обучающей выборке. Расстояние между объектами определяется с использованием выбранной метрики. Метод 1-NN прост в реализации, но может быть вычислительно требователен при больших объемах данных.
2. Метод k-ближайших соседей (k-Nearest Neighbors, k-NN). Этот метод расширяет метод 1-NN. Отличие заключается в количестве ближайших соседей, которые участвуют в классификации объекта [5].
3. Взвешенный k-ближайших соседей (Weighted k-Nearest Neighbors): Этот метод расширяет метод k-NN, присваивая различные веса ближайшим соседям. Веса могут быть определены на основе расстояния или других факторов, чтобы учитывать важность каждого соседа при классификации [6].
4. Метод парзеновского окна (Parzen Window): В этом методе объект классифицируется на основе ядерной функции, которая оценивает плотность объектов каждого класса в окрестности классифицируемого объекта. Ширина окна влияет на важность удаленности объектов при классификации [7].
5. Метод потенциалов (Potential-based Classification): Этот метод основан на определении потенциала для каждого объекта и применении функции потенциала для классификации. Функция потенциала определяет близость объектов и их принадлежность к классам [8].

Проблема исследования заключается в отсутствии четких рекомендаций или наставлений для выбора конкретного метрического метода классификации в конкретной задаче. В области машинного обучения существует широкий спектр метрических методов классификации, таких как метод k-ближайших соседей, методы на основе взвешенного голосования, локальные методы взвешенных соседей и другие. Каждый метод имеет свои сильные и слабые стороны, а также определенные условия применимости.

Для решения данной проблемы необходимо провести сравнительный анализ различных метрических методов классификации в разных сценариях и задачах. Это позволит определить, какие методы лучше справляются с определенными типами данных, объемом выборки, числом классов и особенностями признаков. Кроме того, рекомендации могут учитывать требования к точности классификации, скорости работы и интерпретируемости модели.

Целью курсового проекта является формирование рекомендации по выбору модели.

Для достижения цели необходимо решить следующие задачи.

1. Обзор литературы. Изучить существующие научные статьи, книги и исследования, посвященные метрическим методам классификации. Выявить основные методы и их характеристики, а также рассмотреть примеры исследований, где эти методы применялись в различных задачах.
2. Описание и анализ основных метрических методов классификации
3. Подготовка набора данных. Собрать или использовать доступные наборы данных, соответствующие различным предметным областям. Обеспечить разнообразие данных в терминах признаков, классов, объема выборки и сложности задачи.
4. Реализация и применение каждого из методов для решения задач машинного обучения на различных наборах данных.
5. Анализ результатов и сравнение эффективности каждого метода на различных наборах данных. Определить подходящие метрики оценки качества классификации и провести серию экспериментов, сравнивая различные метрические методы классификации на подготовленных наборах данных. Сравнивать производительность методов в терминах точности, полноты, F-меры.
6. Сделать выводы о применимости каждого метода в различных сценариях и задачах классификации.

Курсовой проект включает два раздела. В первом разделе проведен обзор литературы и алгоритмов по предметной области. Во втором разделе был проведен анализ метрических методов на практике.

# Разбор метрических методов классификации

## Постановка задачи

Пусть на множестве объектов задана функция расстояния Существует целевая зависимость , значения которой известны только на объектах обучающей выборки Множество классов конечно. Требуется построить алгоритм классификации , аппроксимирующий целевую зависимость на всём множестве [4, c. 2].

Другими словами задача классификации заключается в том, чтобы разделить разные ситуации или объекты на группы или классы. Мы имеем некоторые данные об объектах, для которых мы знаем, к каким классам они относятся. Эти данные называются обучающей выборкой. Однако у нас есть и другие объекты, для которых мы не знаем, к каким классам они относятся. Наша задача состоит в разработке алгоритма, который сможет классифицировать любой из этих неизвестных объектов и определить, к какому классу он относится, исходя из имеющихся данных.

## Метрики оценки качества

Для оценки качества понадобятся следующие выкладки:

|  |  |  |  |
| --- | --- | --- | --- |
| Category | | Expert judgments | |
| YES | NO |
| Classifier  Judgement | YES |  |  |
| NO |  |  |

Таблица 1.2.1. The Contingency Table for Category . [9, p. 33]

|  |  |  |  |
| --- | --- | --- | --- |
| Category set | | Expert judgments | |
| YES | NO |
| Classifier  Judgement | YES |  |  |
| NO |  |  |

Таблица 1.2.2 The Global Contingency Table [9, p. 33].

Здесь – Истинно положительный (True Positive), – Ложно Положительный (False Positive), – Ложно Отрицательный (False Negative), – Истинно Отрицательный (True Negative).

### Precision и recall

Precision (точность) измеряет, насколько точно модель классифицирует положительные примеры. Она вычисляется как отношение числа правильно классифицированных положительных примеров к общему числу примеров, которые модель отнесла к положительному классу. Для класса :

Recall (полнота) измеряет, насколько хорошо модель обнаруживает все положительные примеры. Она вычисляется как отношение числа правильно классифицированных положительных примеров к общему числу положительных примеров в данных. Для класса :

Таким образом, precision полезна в задачах, где важно минимизировать ложно положительные результаты, а recall полезна в задачах, где важно минимизировать ложно отрицательные результаты.

Выбор между этими двумя метриками зависит от исходной задачи.

Одним из примеров, когда стоит предпочесть одну метрику другой, может быть задача классификации медицинских изображений для обнаружения редких заболеваний. В таком случае, полнота (recall) может иметь большее значение, поскольку нежелательно пропустить даже один положительный случай.

Напротив, если классификация используется для фильтрации спама в электронной почте, точность (precision) может иметь большее значение. В этом случае, важно минимизировать количество ложно положительных предсказаний, чтобы не пропустить важные сообщения, помечая их как спам.

### метрика и ее частный случай метрика

Является объединением Precision и Recall, показывает приоритет одной метрики над другой. При метрики учитываются в равной степени. [9, p. 33, 36]

-мера полезна если данные несбалансированны, так как она учитывает и точность, и полноту, что позволяет получить общую оценку производительности модели, не учитывая смещение в сторону преобладающего класса.

### метрика и ее частный случай метрика

Accuracy вычисляется как отношение числа правильно классифицированных примеров к общему числу примеров:

[10, стр. 71]

Accuracy может быть полезна, когда классы в данных сбалансированы, то есть количество примеров каждого класса примерно одинаково. В таких случаях она дает общую оценку производительности модели.

## Вычислительная сложность рассматриваемых алгоритмов

Предположим, у нас есть обучающая выборка с N объектами, каждый из которых имеет d признаков, и тестовая выборка с M объектами, которые мы хотим классифицировать или предсказать их значения. Также предположим, что для поиска ближайших соседей используется простой перебор всех объектов.

Для каждого объекта из тестовой выборки мы должны найти k ближайших соседей в обучающей выборке. Сложность этого этапа зависит от выбранного алгоритма поиска ближайших соседей и может быть разной.

Простой перебор. В наивном подходе, где перебираются все объекты обучающей выборки, сложность составляет так как нам нужно сравнить каждый объект из тестовой выборки с каждым объектом из обучающей выборки.

Использование структур данных, таких как k-d деревья или шаровые деревья: В этом случае сложность может быть снижена до или даже если используется приближенный метод поиска ближайших соседей.

Таким образом, общая вычислительная сложность метода k-NN может быть представлена как для простого перебора или для использования структур данных, учитывающих оптимизации поиска ближайших соседей или даже для приближенных алгоритмов.

В методе потенциальных функций для нахождения потенциалов используется алгоритм с вычислительной сложностью , где обозначает количество итераций в алгоритме, а обозначает сложность вычисления потенциала для одного объекта. может быть равно для простого перебора, для использования структур данных, учитывающих оптимизации поиска ближайших соседей или для приближенных алгоритмов.

## Метод ближайшего соседа

### Формулировка

Самый простой из метрических методов. Он относит классифицируемый объект к тому классу, которому принадлежит ближайший обучающий объект:

[4, c. 3-4]

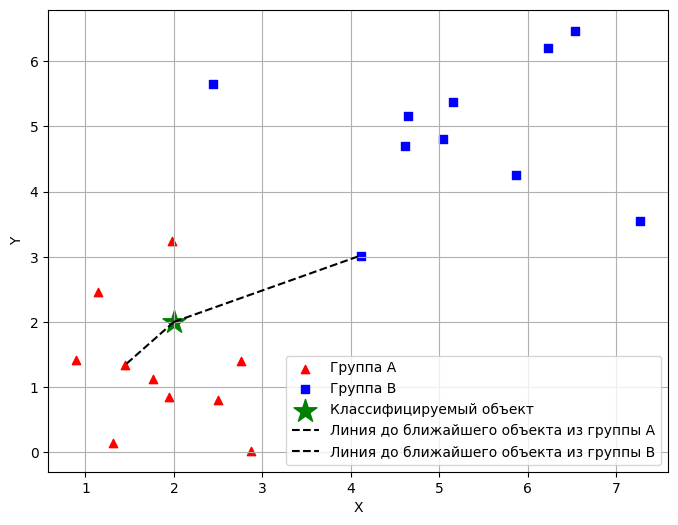


Рисунок 1.1 Визуализация метода ближайшего соседа. Классифицируемый объект следует отнести к классу A, так как ближайший к нему объект относится к этому классу.

### Достоинства и недостатки

Обучение сводится к запоминанию выборки . Единственное достоинство этого алгоритма – простота реализации. Недостатков гораздо больше:

* Неустойчивость к погрешностям. Если среди обучающих объектов есть выброс объект, находящийся в окружении объектов чужого класса, то не только он сам будет классифицирован неверно, но те окружающие его объекты, для которых он окажется ближайшим, также будут классифицированы неверно.
* Отсутствие параметров, которые можно было бы настраивать по выборке. Алгоритм полностью зависит от того, насколько удачно выбрана метрика ρ.
* В результате – низкое качество классификации [4, c. 3-4].

### Алгоритм

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | Алгоритм метода ближайшего соседа:  Вход:  - trainSet: Обучающая выборка с N объектами  - testSet: Тестовая выборка с M объектами  - K: Количество классов  Выход:  - Предсказанные классы или значения для объектов из testSet    Для каждого объекта testObject из testSet:  Найти ближайший объект trainObject из trainSet к testObject:  minDistance = бесконечность  closestObject = None  Для каждого объекта trainObject из trainSet:  distance = Расстояние между testObject и trainObject  Если distance < minDistance:  minDistance = distance  closestObject = trainObject    Предсказать класс testObject на основе класса closestObject  Вернуть предсказанные классы или значения для объектов |

## Метод k ближайших соседей

### Формулировка

Метод ближайших соседей заключается в том, что каждый из соседей голосует за отнесение объекта к своему классу . Алгоритм относит объект u к тому классу, который наберёт большее число голосов:

[4, стр. 4]

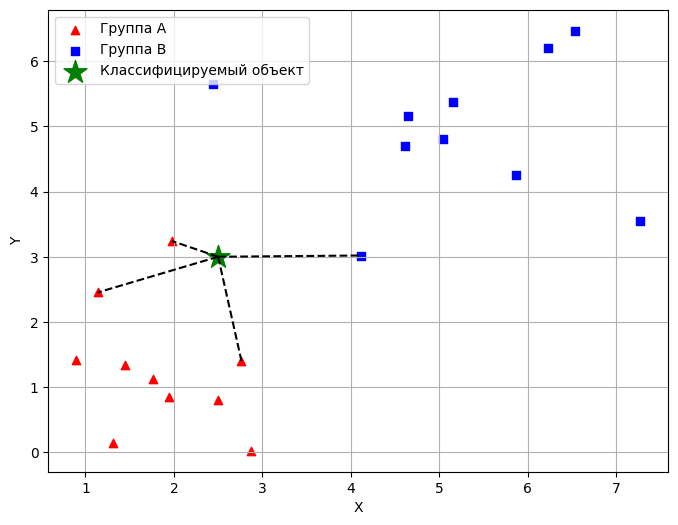


Рисунок 1.2 Визуализация метода k ближайших соседей. В качестве параметра k взято 4 соседа. Объект следует отнести к классу A, так как среди 4 его соседей 3 относится к этому классу, и лишь один объект относится к классу B.

### Алгоритм

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | Алгоритм взвешенных ближайших соседей:  Вход:  - trainSet: Обучающая выборка с N объектами  - testSet: Тестовая выборка с M объектами  - k: Количество соседей  Выход:  - Предсказанные классы или значения для объектов из testSet    Для каждого объекта testObject из testSet:  Найти k ближайший объектов trainObjects из trainSet к testObject:    class\_count = Cловарь, ключ - класс, значение - кол-во объектов  Для каждого объекта trainObject из trainObjects:  distance = Расстояние между testObject и trainObject  class\_count[trainObject.class] += 1    Предсказать класс testObject как наиболее встречающийся класс  Вернуть предсказанные классы для объектов |

### Достоинства и недостатки

Достоинства:

* Простота реализации

Недостатки:

* Приходится хранить обучающую выборку целиком
* Поиск ближайшего соседа предполагает сравнение классифицируемого объекта со всеми объектами выборки за операций. Для задач с большими выборками или высокой частотой запросов это может оказаться накладно. Проблема решается с помощью эффективных алгоритмов поиска ближайших соседей, требующих в среднем операций.
* В простейших случаях метрические алгоритмы имеют крайне бедный набор параметров, что исключает возможность настройки алгоритма по данным.

[4, с. 5]

## Метод взвешенных k ближайших соседей

### Формулировка

Недостаток kNN в том, что максимальная сумма голосов может достигаться на нескольких классах одновременно. В задачах с двумя классами этого можно избежать, если брать только нечётные значения k. Более общая тактика, которая годится и для случая многих классов – ввести строго убывающую последовательность вещественных весов , задающих вклад i-го соседа в классификацию:

Критерий выбора последовательности является эвристикой. [4, с. 4]

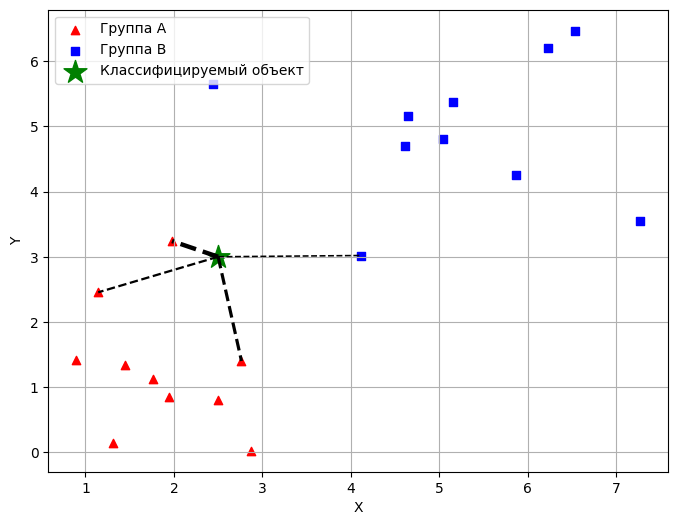


Рисунок 1.3 Иллюстрация метода взвешенных ближайших соседей. Чем толще линия, тем больший вклад вносит сосед в классификацию объекта.

### Алгоритм

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | Алгоритм взвешенных ближайших соседей:  Вход:  - trainSet: Обучающая выборка с N объектами  - testSet: Тестовая выборка с M объектами  - k: Количество соседей  Выход:  - Предсказанные классы или значения для объектов из testSet    Для каждого объекта testObject из testSet:  Найти k ближайший объектов trainObjects из trainSet к testObject:    class\_count = Cловарь, ключ - класс, значение - сумма весов объектов  Для каждого объекта trainObject из trainObjects:  distance = Расстояние между testObject и trainObject  class\_count[trainObject.class] += Вес(distance)    Предсказать класс testObject как наиболее весомый класс  Вернуть предсказанные классы для объектов  Вернуть предсказанные классы для объектов |

### Достоинства и недостатки

Метод имеет достоинства и недостатки метода k ближайших соседей, но добавляет возможность введения различных модификаций весовой функции, что является преимуществом.

## Метод парзеновского окна фиксированной ширины

### Формулировка

Ещё один способ задать веса соседям – определить как функцию от расстояния а не от ранга соседа. Введём функцию ядра невозрастающую на , и рассмотрим алгоритм

Параметр h называется шириной окна и играет примерно ту же роль, что и число соседей k [4, c. 5].

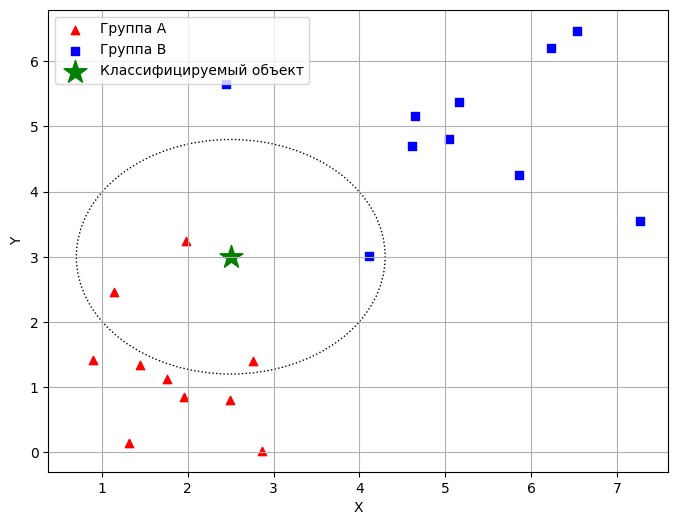


Рисунок 1.4 Иллюстрация метода парзеновского окна фиксированной ширины. При использовании финитного ядра (функция принимает нулевое значение за пределами некоторого компакта), таким образом объекты за пределами параметра h (ширина окна) не вносят свой вклад в классификацию объекта. При использовании нефинитных все объекты принимают участие в классификации, но чем дальше объект, тем меньший вклад он вносит.

### Алгоритм

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | Алгоритм взвешенных ближайших соседей:  Вход:  - trainSet: Обучающая выборка с N объектами  - testSet: Тестовая выборка с M объектами  - h: Ширина окна  Выход:  - Предсказанные классы или значения для объектов из testSet    Для каждого объекта testObject из testSet:  Найти k ближайший объектов trainObjects из trainSet к testObject:    class\_count = Cловарь, ключ - класс, значение - сумма вкладов объектов  Для каждого объекта trainObject из trainObjects:  distance = Расстояние между testObject и trainObject  class\_count[trainObject.class] += Ядро(distance/h)    Предсказать класс testObject как класс с наибольшим вкладом  Вернуть предсказанные классы для объектов |

### Преимущества и недостатки

Основное преимущество, которое дает метод парзеновского окна переменной ширины – это способность к учету локальной плотности. Он может хорошо справляться с областями, где данные плотно сгруппированы и имеют четкую границу.

Из недостатков можно выделить:

* Чувствительность к выбору ширины окна. При использовании фиксированной ширины окна метод становится чувствительным к выбору этой ширины. Неправильно выбранная ширина окна может привести к недооценке или переоценке плотности и, как следствие, к неправильным классификационным результатам. Проблема может быть решена, если применить метод парзеновского окна фиксированной ширины.
* Проблема с выбросами. Метод парзеновского окна фиксированной ширины может быть чувствителен к выбросам. Он не всегда хорошо справляется с областями, где есть выбросы или разреженные области данных.

## Метод парзеновского окна переменной ширины ширины

### Формулировка

Возьмём финитное ядро – невозрастающую функцию положительную на отрезке , и равную нулю вне его. Определим h как наибольшее число, при котором ровно ближайших соседей объекта u получают ненулевые веса Тогда алгоритм принимает вид

Выбор ядра K слабо влияет на качество классификации. На практике ядро либо задаётся априори, либо выбирается по скользящему контролю из нескольких стандартных ядер [4, с. 6].

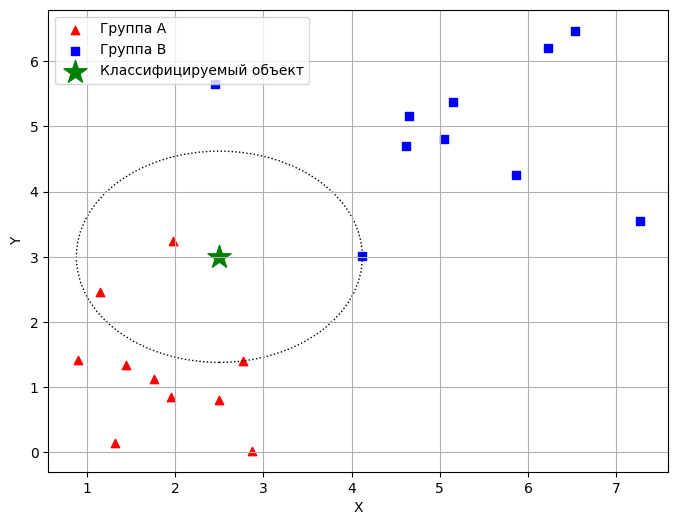


Рисунок 1.5 Иллюстрация метода парзеновского окна переменной ширины. Окружность вокруг классифицируемого объекта включает k=4 объекта, которые будут участвовать в его классификации.

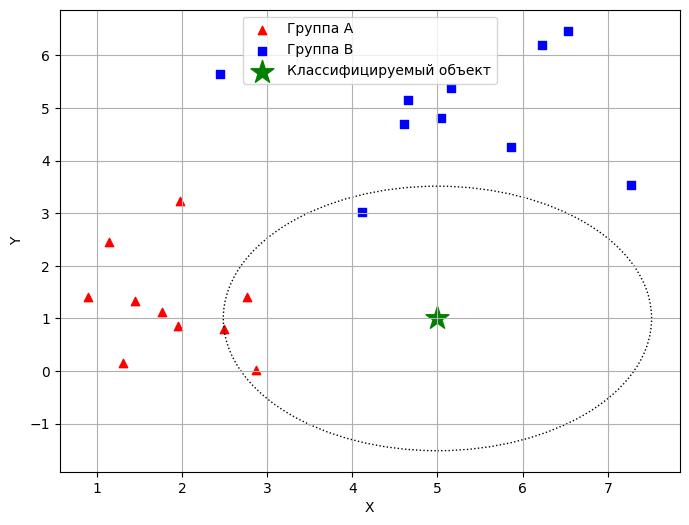


Рисунок 1.6 Иллюстрация метода парзеновского окна переменной ширины. Окружность вокруг другого классифицируемого объекта стала больше, чтобы включить k=4 объекта.

### Алгоритм

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | Алгоритм взвешенных ближайших соседей:  Вход:  - trainSet: Обучающая выборка с N объектами  - testSet: Тестовая выборка с M объектами  - h: Ширина окна  Выход:  - Предсказанные классы или значения для объектов из testSet    Для каждого объекта testObject из testSet:  Найти k+1 ближайший объектов trainObjects из trainSet к testObject:    h = trainObjects[-1]  trainObjects.pop()  class\_count = Cловарь, ключ - класс, значение - сумма вкладов объектов  Для каждого объекта trainObject из trainObjects:  distance = Расстояние между testObject и trainObject  class\_count[trainObject.class] += Ядро(distance/h)    Предсказать класс testObject как класс с наибольшим вкладом  Вернуть предсказанные классы для объектов |

### Преимущества и недостатки

Метод имеет те же преимущества, что и метод парзеновского окна фиксированной ширины, но при этом решает перечисленные в этой работе его недостатки, а именно:

* Чувствительность к выбору ширины окна.
* Проблема с выбросами.

## Метод потенциальных функций

### Формулировка

Допустим теперь, что ядро помещается в каждый обучающий объект и “притягивает” объект u к классу , если он попадает в его окрестность радиуса :

[4, c. 6].

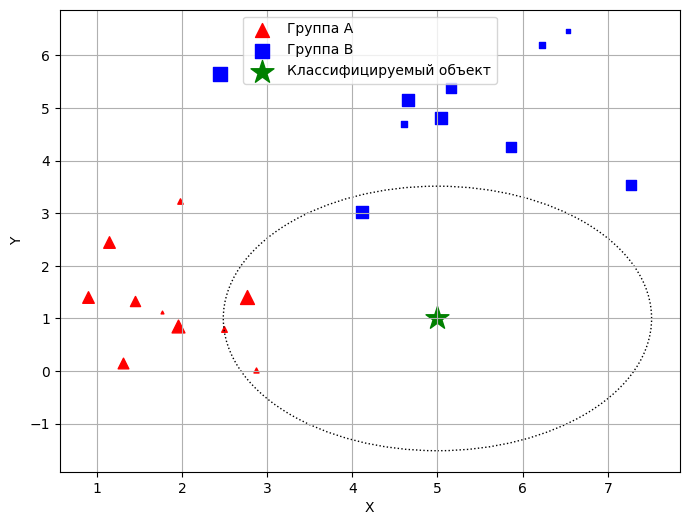


Рисунок 1.7 Иллюстрация метода потенциальных функций. Чем больше объект, тем больше его вклад (потенциал) в классификацию. Ширина окна изображена в виде окружности вокруг классифицируемого объекта.

### Алгоритм

Поиск потенциалов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | Алгоритм ПоискПотенциалов:  Вход:  - X\_train: Обучающая выборка с N объектами  - y\_train: Метки классов для обучающей выборки  - dist\_matrix: Матрица расстояний между объектами  - eps: Пороговое значение ошибки  Выход:  - Потенциалы для объектов обучающей выборки  Инициализация:  potentials = Заполнить массив длиной N нулями  err = 1.0  Пока err > eps:  Пока True:  rand = Случайное целое число в диапазоне от 0 до N-1  cl = ПолучитьБлижайшегоСоседа(y\_train, dist\_matrix, rand)  Если cl != y\_train[rand]:  potentials[rand] += 1  Прервать цикл  predict = Предсказать(X\_train, potentials, dist\_matrix)  err = 1 - Точность(predict, y\_train)  Вернуть potentials  Алгоритм: |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | Алгоритм взвешенных ближайших соседей:  Вход:  - trainSet: Обучающая выборка с N объектами  - testSet: Тестовая выборка с M объектами  - h: Набор ширин окон для testSet  - potencials: Потенциалы объектов из trainSet  Выход:  - Предсказанные классы или значения для объектов из testSet    Для каждого объекта testObject из testSet:  Найти k ближайший объектов trainObjects из trainSet к testObject:    class\_count = Cловарь, ключ - класс, значение - сумма вкладов объектов  Для каждого объекта trainObject из trainObjects:  distance = Расстояние между testObject и trainObject  class\_count[trainObject.class] += Ядро(distance/h[i])\*potencials[i]    Предсказать класс testObject как класс с наибольшим вкладом  Вернуть предсказанные классы для объектов |

### Преимущества и недостатки

Преимущества:

* Устойчивость к шуму: Метод потенциальных функций обычно имеет хорошую устойчивость к шуму и выбросам в данных. Поскольку он основан на поиске потенциалов для объектов, он может лучше адаптироваться к неправильно классифицированным объектам и корректировать их значения.
* Возможность учета весов: Метод потенциальных функций может учитывать веса объектов в обучающей выборке. Это позволяет отдавать большее влияние на определенные объекты и улучшает качество классификации.

Недостатки:

* Вычислительная сложность. Метод потенциальных функций может быть вычислительно сложным, особенно при большом размере обучающей выборки и высокой размерности пространства признаков. Подсчет потенциалов для каждого объекта может требовать значительных вычислительных ресурсов.
* Неустойчивость к перекрытию классов. Метод потенциальных функций может иметь проблемы с классификацией в случае, когда классы сильно перекрываются. В таких случаях может быть сложно правильно определить потенциалы и достичь высокой точности классификации.

## Выводы по разделу

Метрические методы классификации представляют собой широко используемый класс алгоритмов машинного обучения, который основан на измерении расстояний или сходства между объектами. Они являются простыми и интуитивно понятными, поэтому пользуются популярностью во многих областях, включая распознавание образов, биометрию, анализ текста.

В данной главе были разобраны пять метрических методов классификации. Каждый из них имеет свои преимущества и недостатки, поэтому важно выбирать метод в зависимости от поставленной задачи. Кроме того, для полного понимания и оценки этих методов необходимо провести практические эксперименты, чтобы проверить их преимущества и ограничения.

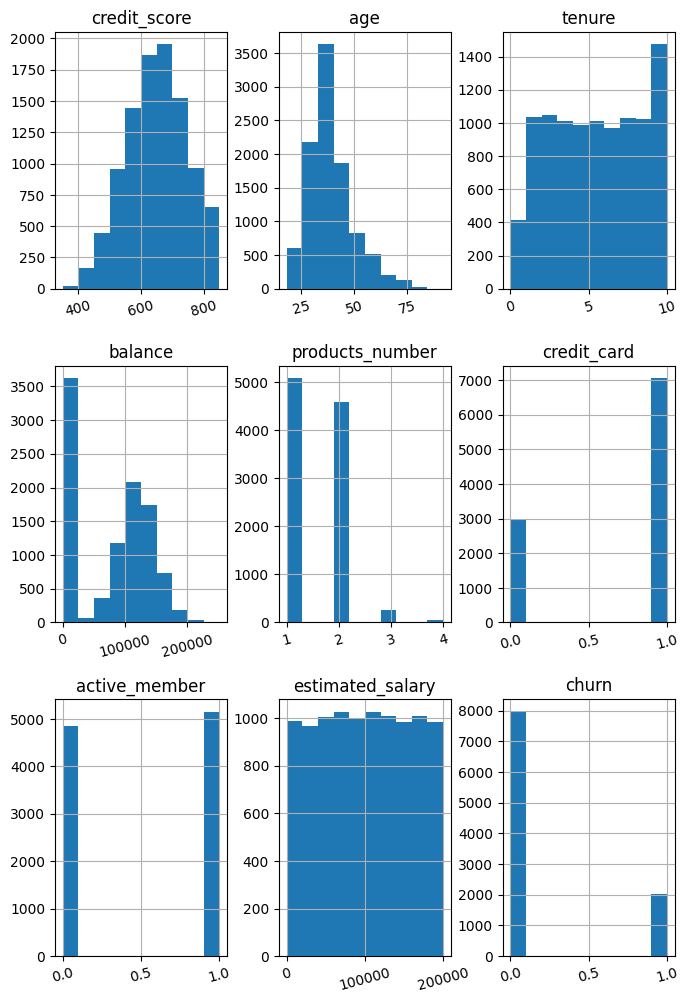
Таким образом, изучение метрических методов классификации позволяет получить полезные инструменты для анализа и классификации данных. Однако для достижения наилучших результатов необходимо тщательно выбирать и применять соответствующий метод в каждой конкретной задаче.

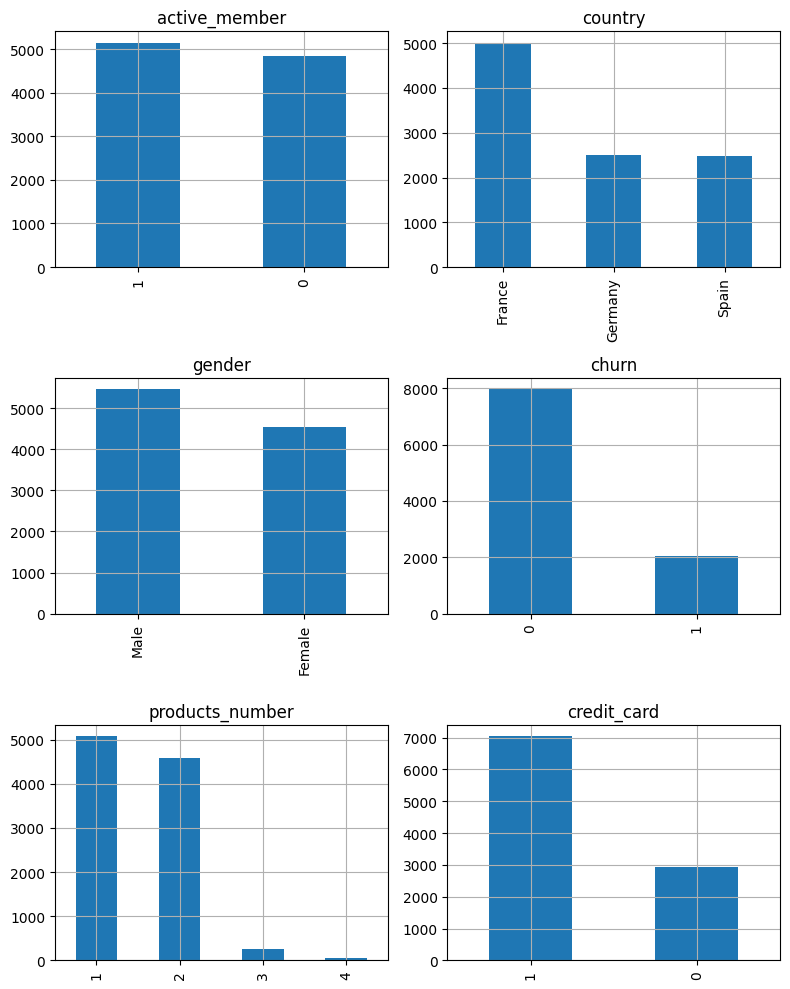
# Применение алгоритмов на практике

## Bank Customer Churn Dataset[[1]](#footnote-1)

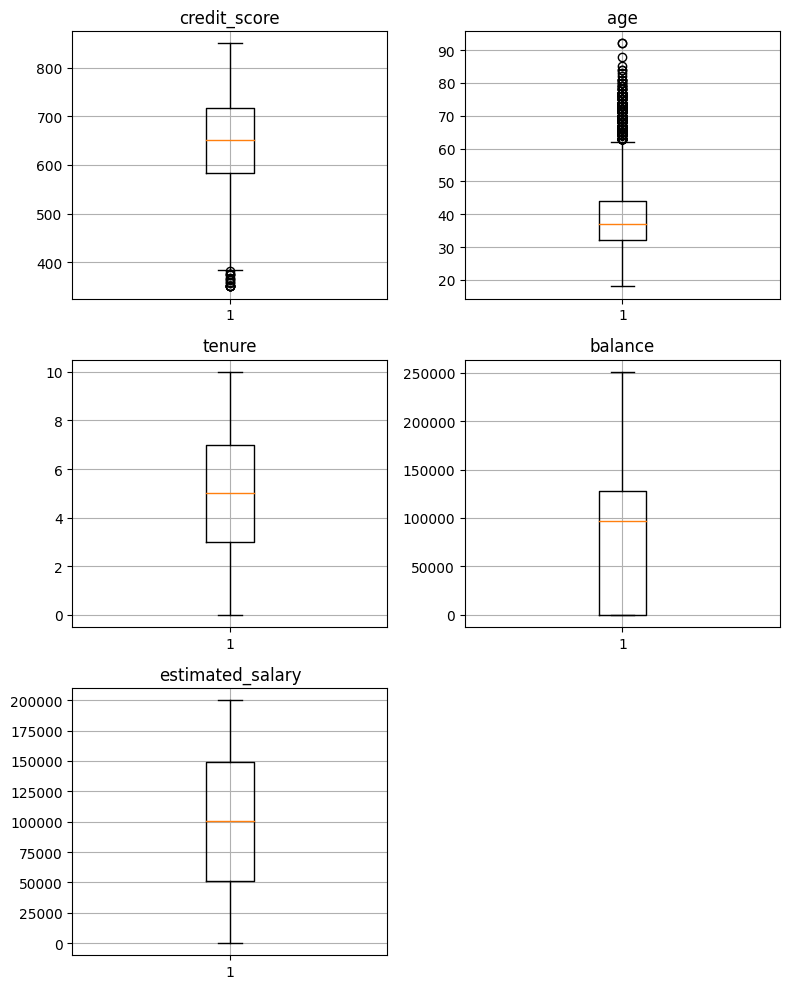
1. customer\_id, идивидуальный номер
2. credit\_score, кредитный рейтинг
3. country, страна проживания
4. gender, пол
5. age, возраст
6. tenure, Со скольки лет имеет банковский счет в ABC Bank
7. balance, баланс
8. products\_number, номер продукта из банка
9. credit\_card, У этого клиента есть кредитная карта?
10. active\_member, Является ли он/она активным членом банка?
11. estimated\_salary, Заработная плата владельца счета
12. churn, используется в качестве целевой переменной. 1, если клиент выходил из банка в течение какого-то периода, или 0, если нет.

Распределение данных показано на следующих рисунках:





По графику churn хорошо виден дисбаланс данных. Для алгоритма ближайших соседей лучше дополнить недостающий класс искусственно сгенерированными данными с помощью алгоритма SMOTE. Точность моделей повысилась после применения SMOTE.



Графики выше иллюстрируют, что возможно в колонках credit\_score и age есть аномальные значения – это нужно проверить.

Действительно, в колонке age 133 значения выходят за диапазон 3-х сигм, но эти значения не стоит выбрасывать, так как они все еще входят разумные пределы до 90 лет.

В колонке credit\_score по правилу трех сигм найдено всего 8 значений, которые почти попадают в диапазон неаномальных значений, поэтому их тоже не следует отбрасывать.

Корреляционная матрица не показала значимого коррелирования признаков.

Результаты работы алгоритмов представлены в таблице ниже.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1** | **Precision** | **Recall** |
| **KNN** | 0.7753 | **0.4871** | **0.4482** | 0.5333 |
| **Weight KNN** | 0.7753 | **0.4871** | **0.4482** | 0.5333 |
| **Fixed-width Parzen Window** | 0.7093 | 0.4591 | 0.3656 | **0.6167** |
| **Variable-width Parzen Window** | 0.7440 | 0.4754 | 0.4028 | 0.5800 |
| **Potential Function** | **0.7980** | 0.0066 | 0.2000 | 0.0033 |
| **Sk-learn** | 0.7753 | **0.4871** | **0.4482** | 0.5333 |

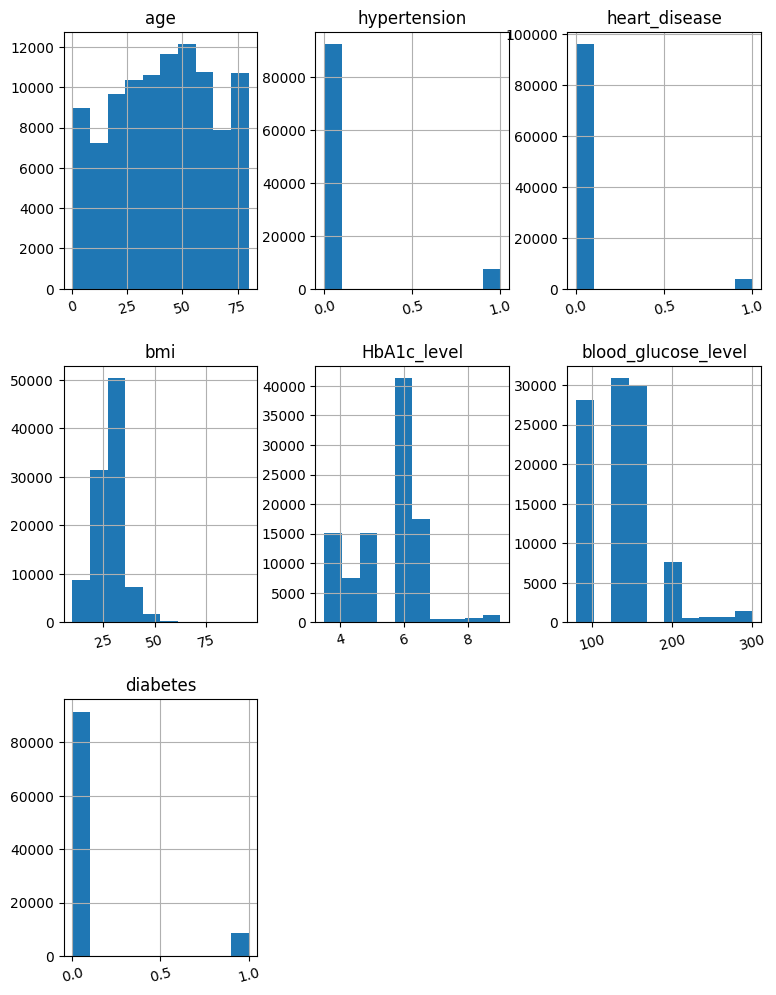
* Наибольшие показатели основной метрики (F1) у алгоритма knn и knn-wieghted.
* Метод потенциальных функций показал наибольшее accuracy, но базовый классификатор выдает 0.7753, что говорит о низком уровне качества модели.
* Стоит отметить, что Fixed-width Parzen Window имеет наибольший показатель Recall.

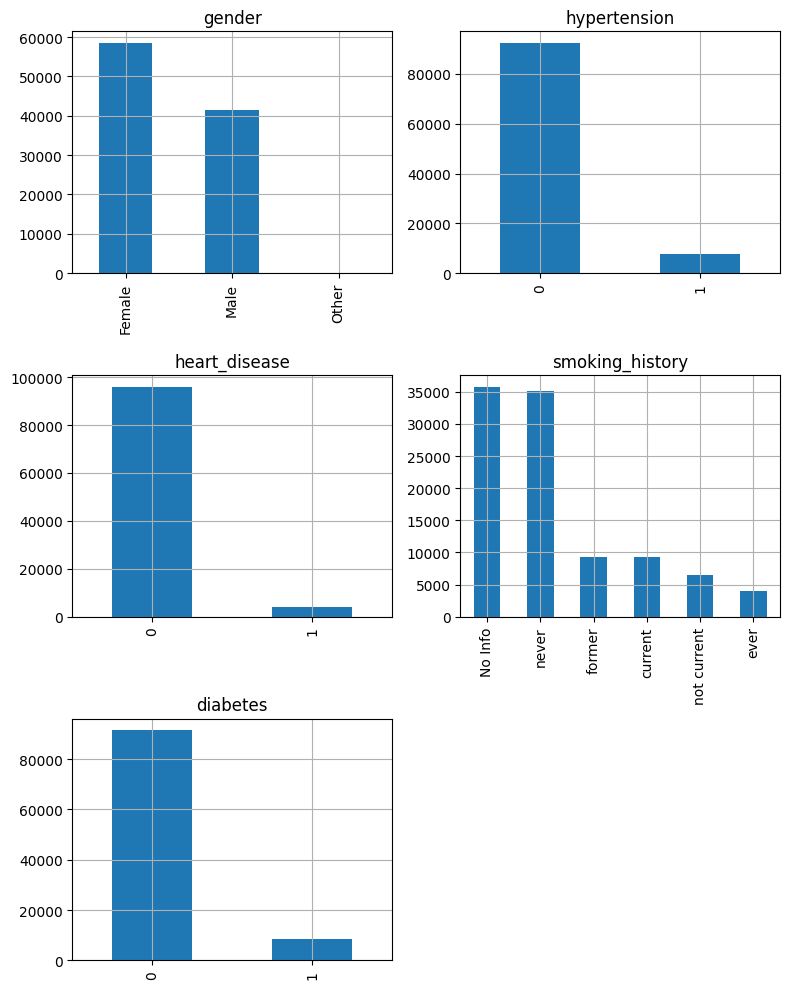
## Diabetes prediction dataset[[2]](#footnote-2)

Набор данных прогнозирования диабета представляет собой набор медицинских и демографических данных пациентов, а также их статус диабета (положительный или отрицательный). Данные включают такие характеристики, как возраст, пол, индекс массы тела (ИМТ), гипертония, болезни сердца, история курения, уровень HbA1c и уровень глюкозы в крови. Этот набор данных можно использовать для создания моделей машинного обучения для прогнозирования диабета у пациентов на основе их истории болезни и демографической информации. Это может быть полезно медицинским работникам при выявлении пациентов, которые могут подвергаться риску развития диабета, и при разработке индивидуальных планов лечения. Кроме того, набор данных может быть использован исследователями для изучения взаимосвязей между различными медицинскими и демографическими факторами и вероятностью развития диабета.

1. gender, Пол относится к биологическому полу человека, который может влиять на его восприимчивость к диабету. В нем есть три категории: мужские, женские и другие.
2. age, Возраст является важным фактором, поскольку диабет чаще диагностируется у пожилых людей. Возраст в нашем наборе данных варьируется от 0 до 80 лет.
3. Hypertension, Гипертония — это заболевание, при котором артериальное давление постоянно повышено. Оно имеет значения 0 или 1, где 0 означает, что у них нет гипертонии, а 1 означает, что у них есть гипертония.
4. heart\_disease, Сердечно-сосудистые заболевания — еще одно заболевание, связанное с повышенным риском развития диабета. Оно имеет значения 0 или 1, где 0 означает, что у них нет болезни сердца, а 1 означает, что у них есть болезнь сердца.
5. smoking\_history, Курение также считается фактором риска развития диабета и может усугубить осложнения, связанные с диабетом.
6. Bmi, ИМТ (индекс массы тела) — это показатель содержания жира в организме, основанный на весе и росте. Более высокие значения ИМТ связаны с более высоким риском развития диабета. Диапазон ИМТ в наборе данных составляет от 10,16 до 71,55. ИМТ менее 18,5 — недостаточный вес, 18,5–24,9 — нормальный, 25–29,9 — избыточный вес, 30 и более — ожирение.
7. HbA1c\_level, Уровень HbA1c (гемоглобина A1c) — это показатель среднего уровня сахара в крови человека за последние 2-3 месяца. Более высокие уровни указывают на больший риск развития диабета. В основном более 6,5% уровня HbA1c указывает на диабет.
8. blood\_glucose\_level, Уровень глюкозы в крови относится к количеству глюкозы в кровотоке в данный момент времени. Высокий уровень глюкозы в крови является ключевым показателем диабета.
9. Diabetes, Диабет является прогнозируемой целевой переменной, где значения 1 указывают на наличие диабета, а 0 — на отсутствие диабета.

Распределения данных представлены ниже.





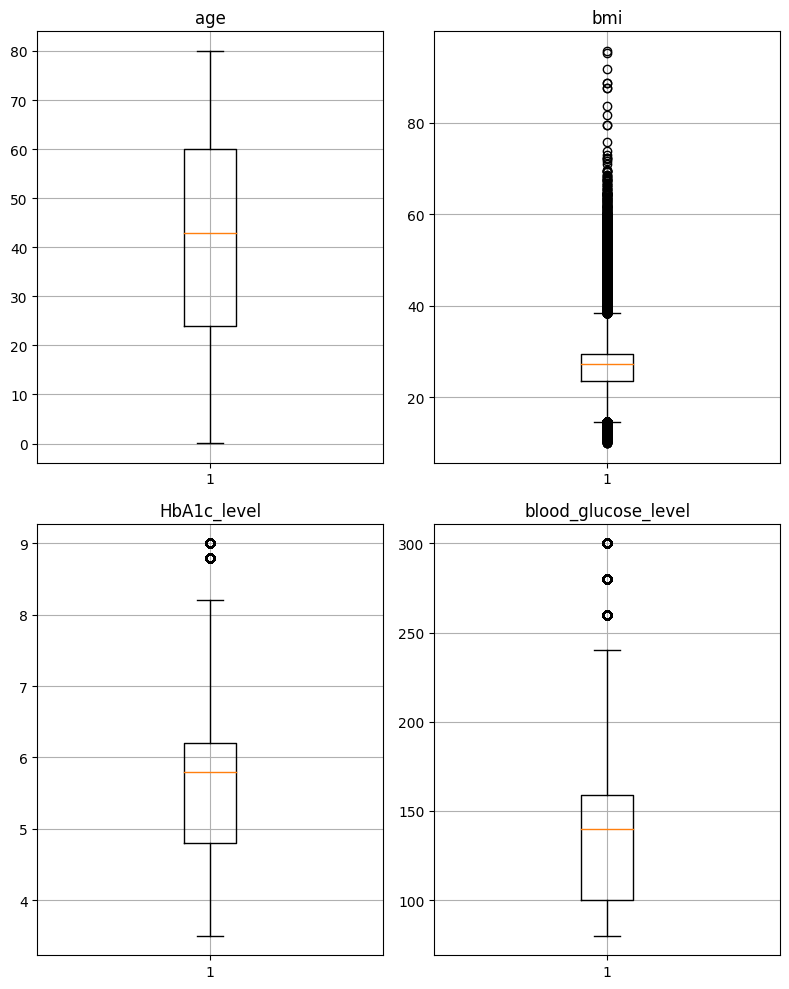


Рисунок выше демонстрирует, что в колонках bmi, blood\_glucose\_level, HbA1c\_level присутствуют экстремальные значения, но при анализе было принято решение не отбрасывать эти данные.

Так как данные несбалансированны, было принято уменьшить преобладающий класс с помощью алгоритма NearMiss. Точность была несколько повышена.

Итоговые результаты работы алгоритма представлены в таблице ниже.

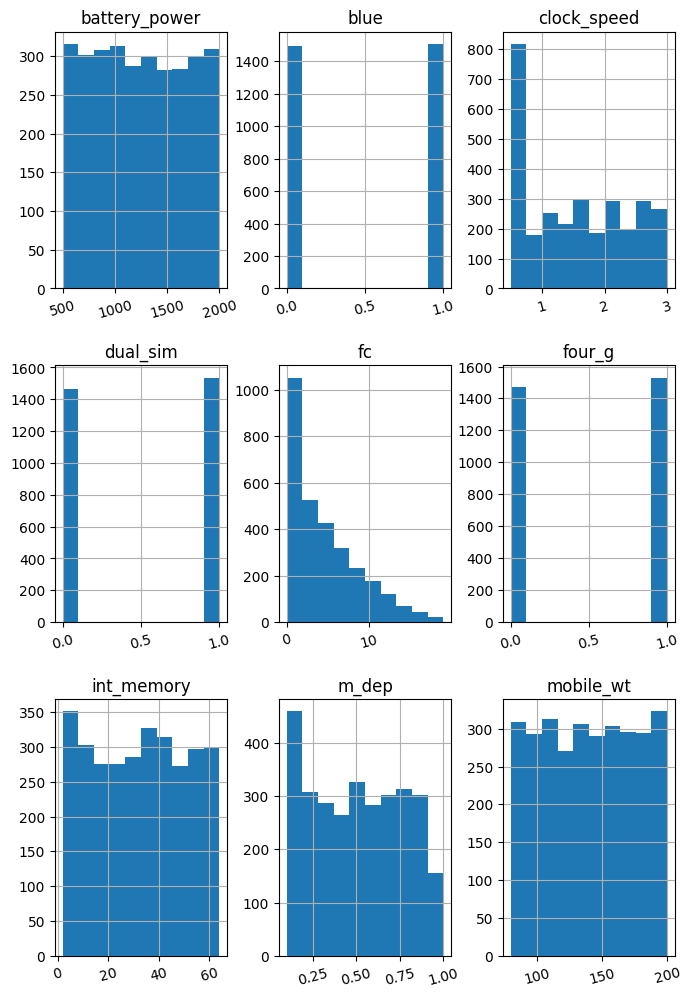
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1** | **Precision** | **Recall** |
| **KNN** | 0.8454 | **0.4403** | **0.3191** | 0.7097 |
| **Weight KNN** | 0.8357 | 0.4304 | 0.3061 | **0.7245** |
| **Fixed-width Parzen Window** | 0.8349 | 0.4051 | 0.293 | 0.6560 |
| **Variable-width Parzen Window** | 0.8436 | 0.4374 | **0.3161** | 0.7097 |
| **Potential Function** | **0.9131** | 0.0046 | 0.1250 | 0.0023 |
| **Sk-learn** | 0.8454 | **0.4403** | **0.3191** | 0.7097 |

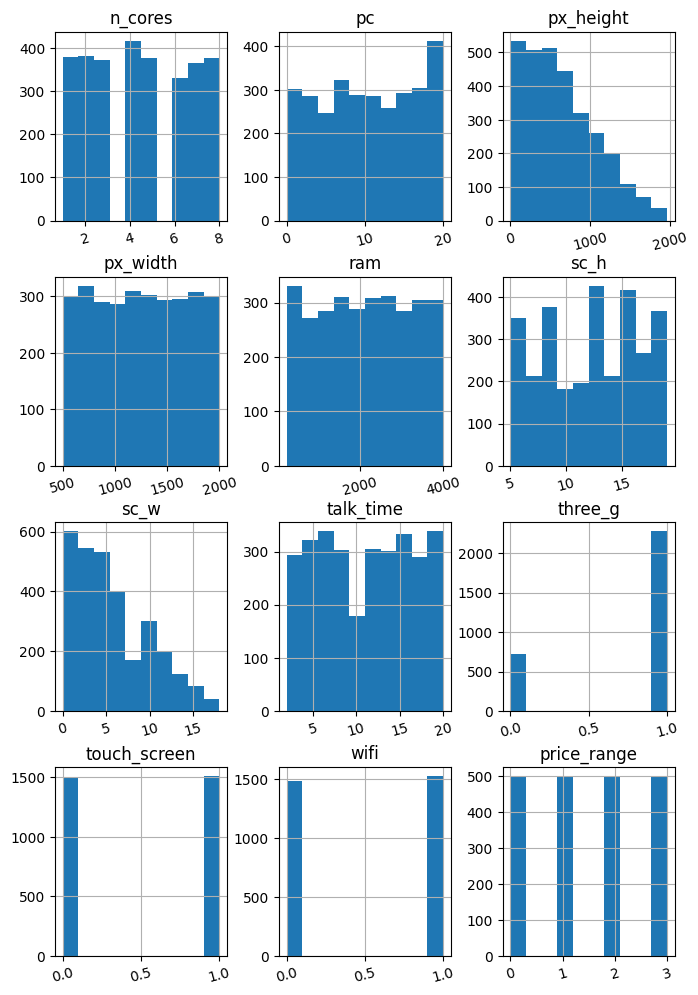
* Наибольшие показатели основной метрики (F1) у алгоритма knn.
* Метод потенциальных функций показал наибольшее accuracy, но базовый классификатор выдает 0.9143, что говорит о низком уровне качества модели.
* Стоит отметить, что knn-wieghted имеет наибольший показатель Recall.

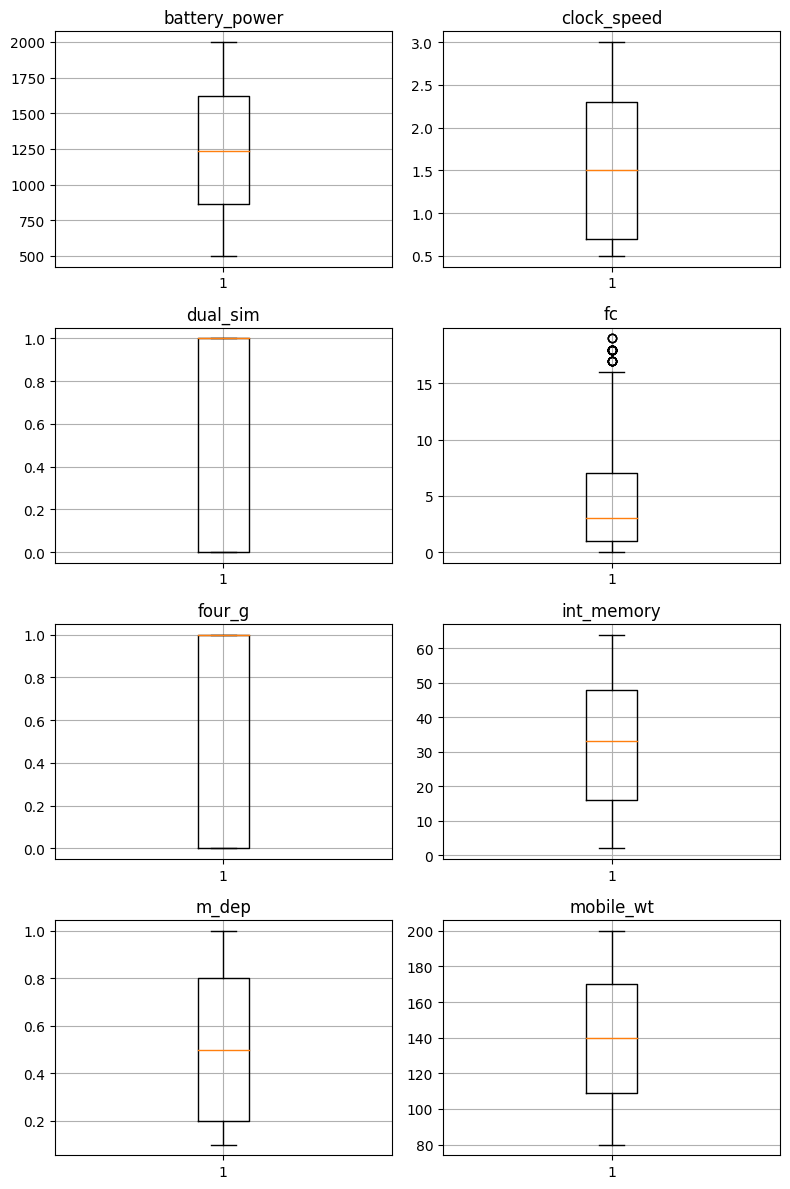
## Mobile Price Classification[[3]](#footnote-3)

1. battery\_power, суммарная энергия, которую батарея может хранить за один раз, измеряется в мАч.
2. blue, есть блютуз или нет
3. clock\_speed, скорость, с которой микропроцессор выполняет инструкции
4. dual\_sim, есть поддержка двух сим карт или нет
5. fc, мегапиксели фронтальной камеры
6. four\_g, есть 4G или нет
7. int\_memory, внутренняя память в гигабайтах
8. m\_dep, толщина устройства
9. mobile\_wt, вес мобильного телефона
10. n\_cores, количество ядер процессора
11. pc, мегапиксели основной камеры
12. px\_height, высота в пикселях
13. px\_width, ширина в пикселях
14. ram, оперативная память в мегабайтах
15. sc\_h, высота в сантиметрах
16. sc\_w, ширина в сантиметрах
17. talk\_time, максимальное время работы от одного заряда батареи, когда вы говорите
18. three\_g, есть 3G или нет
19. touch\_screen, есть сенсорный экран или нет
20. wifi, есть вайфай или нет
21. price\_range, целевая переменная, категория стоимости телефона

Распределение признаков изображено на рисунках ниже.







При проверке подозрительных значений не было выявлено значительных отклонений.

Результаты работы алгоритмов в таблице ниже.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1** | **Precision** | **Recall** |
| **KNN** | 0.5925 | 0.6035 | 0.6499 | 0.5925 |
| **Weight KNN** | **0.6025** | **0.6142** | **0.6587** | **0.6025** |
| **Fixed-width Parzen Window** | 0.5825 | 0.5986 | 0.6492 | 0.5825 |
| **Variable-width Parzen Window** | 0.5950 | 0.6066 | 0.6533 | 0.595 |
| **Potential Function** | 0.3450 | 0.3416 | 0.3647 | 0.3450 |
| **Sk-learn** | **0.6025** | **0.6142** | **0.6587** | **0.6025** |

* Наибольшие показатели основной метрики (F1) у алгоритма knn-wieghted.

## Выводы по разделу

Алгоритмы действительно чувствительны к дисбалансу классов.

В зависимости от задачи могут быть применимы разные методы, но наибольшие показатели в тестах как правило у алгоритма knn, поэтому этот алгоритм стоит пробовать первым. Однако для достижения оптимальной производительности необходимо тщательно настраивать параметры методов и учитывать особенности данных. Более глубокое исследование и сравнение с другими методами классификации могут быть предметом дальнейших исследований.

# Заключение

В ходе курсового проекта были рассмотрены и реализованы алгоритмы: k ближайших соседей, взвешенные k ближайших соседей, метод парзеновского окна фиксированной и переменной ширины, метод потенциальных функций. Кроме того, результат сравнивался с уже готовыми алгоритмами из библиотеки scikit-learn[[4]](#footnote-4). При реализации были использованы библиотеки: numpy[[5]](#footnote-5), pandas[[6]](#footnote-6), matplotlib[[7]](#footnote-7).

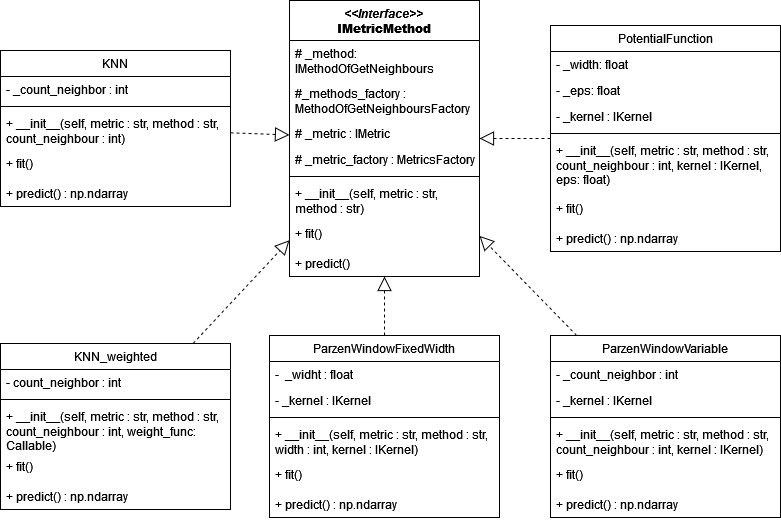
Для оценки качества работы использовался ряд метрик качества: accuracy, precision, recall, F1.

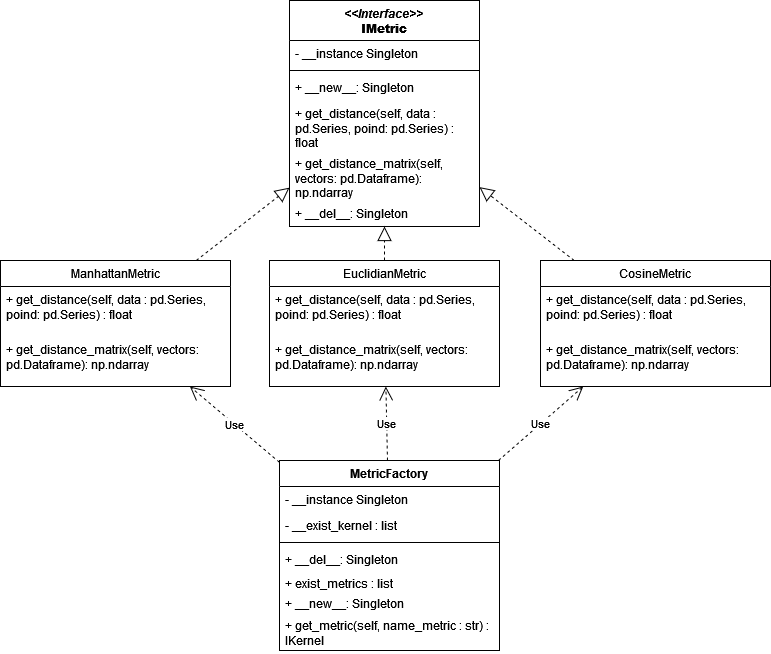
Результаты экспериментов показали, что эффективность метрических методов классификации может сильно зависеть от различных факторов, таких как выбор оптимальных параметров и особенности данных.

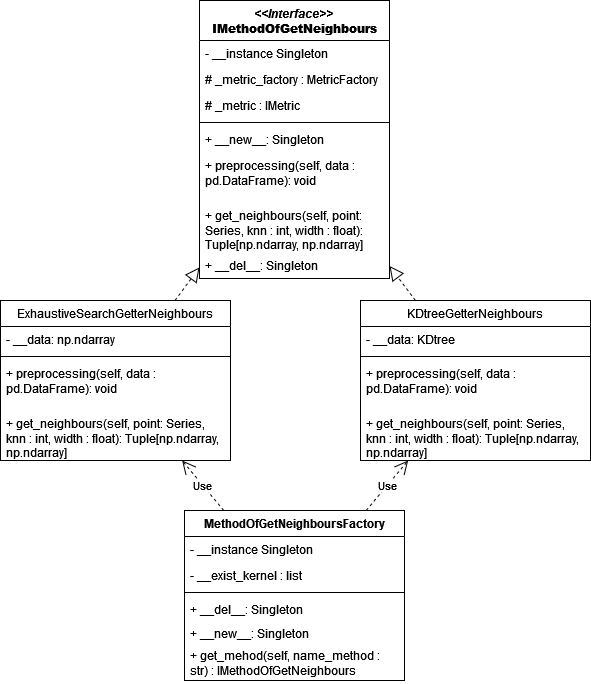
В целом, все методы, кроме метода потенциальных функций, имеют свои преимущества и недостатки

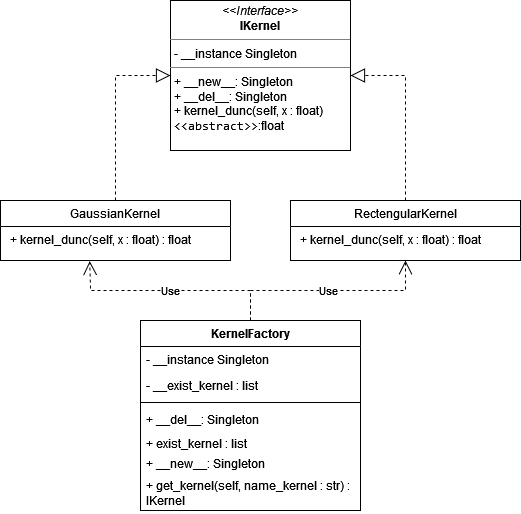
В дальнейшем исследовании можно рассмотреть другие метрические методы классификации, а также провести сравнительный анализ с другими методами машинного обучения. Это поможет лучше понять применимость и эффективность метрических методов в различных задачах классификации.

При реализации были применены различные паттерны программирования: singleton, factory, strategy. Программная реализация представлена в приложении А. Структура программы представлена ниже.









# Библиографический список

Xing W., Bei Y. Medical health big data classification based on KNN classification algorithm //IEEE Access. – 2019. – Т. 8. – С. 28808-28819.

Trstenjak B., Mikac S., Donko D. KNN with TF-IDF based framework for text categorization //Procedia Engineering. – 2014. – Т. 69. – С. 1356-1364.

Zhu Y. et al. Private-knn: Practical differential privacy for computer vision //Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. – 2020. – С. 11854-11862.

Воронцов К. В. Метрические алгоритмы классификации. – 2013.

Peterson L. E. K-nearest neighbor //Scholarpedia. – 2009. – Т. 4. – №. 2. – С. 1883.

Dudani S. A. The distance-weighted k-nearest-neighbor rule //IEEE Transactions on Systems, Man, and Cybernetics. – 1976. – №. 4. – С. 325-327.

Kwak N., Choi C. H. Input feature selection by mutual information based on Parzen window //IEEE transactions on pattern analysis and machine intelligence. – 2002. – Т. 24. – №. 12. – С. 1667-1671.

Alimardani F. et al. Classification of bipolar disorder and schizophrenia using steady-state visual evoked potential based features //IEEE Access. – 2018. – Т. 6. – С. 40379-40388.

Sebastiani F. Machine learning in automated text categorization //ACM computing surveys (CSUR). – 2002. – Т. 34. – №. 1. – С. 1-47.

Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. – Litres, 2022.

# Приложения

## Приложение А. Листинг программы

**Модуль ядер (Kernels.py)**

from abc import ABC, abstractmethod

import numpy as np

\_\_all\_\_ = [

'IKernel',

'GaussianKernel',

'KernelFactory',

'RectangularKernel']

class IKernel(ABC):

\_\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if cls.\_\_instance is None:

cls.\_\_instance = super().\_\_new\_\_(cls)

return cls.\_\_instance

def \_\_del\_\_(self):

IKernel.\_\_instance = None

@abstractmethod

def kernel\_func(self, x : float) -> float:

raise NotImplementedError()

class GaussianKernel(IKernel):

def kernel\_func(self, r: float) -> float:

return 1/np.sqrt(2 \* np.pi) \* np.exp(-2 \* np.power(r, 2))

class KernelFactory():

\_\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if cls.\_\_instance is None:

cls.\_\_instance = super().\_\_new\_\_(cls)

return cls.\_\_instance

def \_\_del\_\_(self):

KernelFactory.\_\_instance = None

\_\_exist\_kernel : list = ["rectangular", "gaussian"]

@property

def exist\_kernel(self) -> list:

return self.\_\_exist\_kernel

def kernel\_exist(self, name : str) -> bool:

return name in self.\_\_exist\_kernel

def get\_kernel(self, name\_kernel : str) -> IKernel:

if name\_kernel == "rectangular":

return RectangularKernel()

elif name\_kernel == "gaussian":

return GaussianKernel()

else:

raise ValueError("Unknown kernel")

class RectangularKernel(IKernel):

def kernel\_func(self, r: float) -> float:

if np.abs(r) <= 1:

return 0.5

else:

return 0

**Модуль получения ближайших соседей (MethodsOfGetNeighbours.py)**

from abc import ABC, abstractmethod

import numpy as np

import pandas as pd

from sklearn.neighbors import KDTree

from .Metrics import \*

\_\_all\_\_ = [

'IMethodOfGetNeighbours',

'ExhaustiveSearchGetterNeighbours',

'KDTreeGetterNeighbours',

'MethodOfGetNeighboursFactory']

class IMethodOfGetNeighbours(ABC):

"""Common interface for getting neighbours of a given point."""

\_metric : str

\_metric\_factory = MetricsFactory()

def \_\_init\_\_(self, metric : str) -> None:

self.\_metric = metric

@abstractmethod

def preprocessing(self, data : pd.DataFrame) -> None:

raise NotImplementedError()

@abstractmethod

def get\_neighbours(self, point : pd.Series, knn : int = -1,

width : float = -1) -> tuple[np.ndarray, np.ndarray]:

"""

if width == -1 and knn != -1 will then return knn nearest neighbors.

if width != -1 and knn == -1 will then return the nearest neighbors

within the radius width.

Other configurations are not possible

Args:

point (pd.Series): classification point

knn (int): count neighbours. Defaults to -1.

width (float): width of window. Defaults to -1.

Returns:

tuple[np.ndarray, np.ndarray]: nearest\_neighbor\_index, distances.

"""

raise NotImplementedError()

class ExhaustiveSearchGetterNeighbours(IMethodOfGetNeighbours):

"""Exhaustive search for neighbours of a given point."""

@property

def data(self):

return self.\_\_data

\_\_data : np.ndarray

def preprocessing(self, data: pd.DataFrame) -> None:

self.\_\_data = np.array(data)

def get\_neighbours(self, point: pd.Series, knn : int = -1,

width : float = -1) -> tuple[np.ndarray, np.ndarray]:

distances = self.\_metric\_factory.get\_metrics(self.\_metric).get\_distance(self.\_\_data, point)

if width == -1 and knn != -1:

nearest\_neighbor\_index = np.argpartition(distances, knn, axis=None)[:knn]

return nearest\_neighbor\_index, distances

elif width != -1 and knn == -1:

index\_elem\_less\_width = np.where(distances < width)[0]

return index\_elem\_less\_width, np.take(distances, index\_elem\_less\_width)

class KDTreeGetterNeighbours(IMethodOfGetNeighbours):

"""Using KDtree for getting neighbours of a given point."""

\_\_kdtree : KDTree

@property

def data(self):

return self.\_\_kdtree

def preprocessing(self, data: pd.DataFrame) -> None:

self.\_\_kdtree = KDTree(data, metric=self.\_metric)

def get\_neighbours(self, point: pd.Series, knn : int = -1,

width : float = -1) -> tuple[np.ndarray, np.ndarray]:

if width == -1 and knn != -1:

distance, nearest\_neighbor\_index = self.\_\_kdtree.query(point.reshape(1, -1), k=knn)

return nearest\_neighbor\_index.ravel(), distance.ravel()

elif width != -1 and knn == -1:

# Define a condition

nearest\_neighbor\_index, distance = self.\_\_kdtree.query\_radius(point.reshape(1, -1), r=width, return\_distance=True)

return nearest\_neighbor\_index.ravel()[0].ravel(), distance.ravel()[0].ravel() # FIXME: I don't know why query\_radius returned np.array([np.array([

class MethodOfGetNeighboursFactory:

"""Use this class to create a method of get neighbours."""

\_\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if cls.\_\_instance is None:

cls.\_\_instance = super().\_\_new\_\_(cls)

return cls.\_\_instance

def \_\_del\_\_(self):

MethodOfGetNeighboursFactory.\_\_instance = None

\_\_exist\_methods : list = ["kdtree", "exhaustive"]

@property

def exist\_metrics(self) -> list:

return self.\_\_exist\_methods

def method\_exist(self, name : str) -> bool:

return name in self.\_\_exist\_methods

def get\_method(self, name\_method : str,

metric : str) -> IMethodOfGetNeighbours:

"""Use this function to create a method of get neighbours.

Args:

name\_method (str): name of the method.

Possible values: kdtree, exhaustive.

metric (str): metric to use.

Returns:

IMethodOfGetNeighbours: method of get neighbours.

"""

if name\_method == "kdtree":

return KDTreeGetterNeighbours(metric)

elif name\_method == "exhaustive":

return ExhaustiveSearchGetterNeighbours(metric)

**Модуль метрик (Metrics.py)**

from abc import ABC, abstractmethod

import numpy as np

import pandas as pd

from scipy.spatial.distance import cdist

\_\_all\_\_ = [

'IMetric',

'MetricsFactory',

'ManhattanMetric',

'EuclideanMetric',

'CosineMetric']

class IMetric(ABC):

\_\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if cls.\_\_instance is None:

cls.\_\_instance = super().\_\_new\_\_(cls)

return cls.\_\_instance

def \_\_del\_\_(self):

IMetric.\_\_instance = None

@abstractmethod

def get\_distance(self, data : pd.Series, point : pd.Series) -> float:

"""

Args:

data (pd.Series): vector1

point (pd.Series): vector2

Returns:

float: distance between data and point

"""

pass

@abstractmethod

def get\_distance\_matrix(self, vectors : pd.DataFrame) -> np.ndarray:

"""

Args:

data (pd.DataFrame): vectors

Returns

-------

np.ndarray: distance matrix

"""

raise NotImplementedError

class ManhattanMetric(IMetric):

def get\_distance(self, data : pd.Series, point : pd.Series) -> float:

return np.sum(np.abs(data - point), axis=-1)

def get\_distance\_matrix(self, vectors : pd.DataFrame) -> np.ndarray:

return cdist(vectors, vectors, metric='cityblock')

class EuclideanMetric(IMetric):

def get\_distance(self, data : pd.Series, point : pd.Series) -> float:

return np.linalg.norm(data - point, axis=-1)

def get\_distance\_matrix(self, vectors : pd.DataFrame) -> np.ndarray:

return cdist(vectors, vectors, metric='euclidean')

class CosineMetric(IMetric):

def get\_distance(self, data : pd.Series, point : pd.Series) -> float:

return (1 - data.dot(point) /

(np.linalg.norm(data, axis=-1) \* np.linalg.norm(point, axis=-1)))

def get\_distance\_matrix(self, vectors : pd.DataFrame) -> np.ndarray:

return cdist(vectors, vectors, metric='cosine')

class MetricsFactory:

"""Use to create metrics"""

\_\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if cls.\_\_instance is None:

cls.\_\_instance = super().\_\_new\_\_(cls)

return cls.\_\_instance

def \_\_del\_\_(self):

MetricsFactory.\_\_instance = None

\_\_exist\_metrics : list = ["euclidean", "cityblock", 'cosine']

@property

def exist\_metrics(self) -> list:

return self.\_\_exist\_metrics

def metrics\_exist(self, name : str) -> bool:

return name in self.\_\_exist\_metrics

def get\_metrics(self, name : str) -> IMetric:

"""

Use to create metrics

Args:

name (str): name of metric

Exists metrics: euclidean, cityblock, cosine

Returns:

IMetric: class of metric

"""

if name == "euclidean":

return EuclideanMetric()

elif name == "cityblock":

return ManhattanMetric()

elif name == 'cosine':

return CosineMetric()

**Модуль метрических методов (MetricMethod.py)**

from abc import abstractmethod

from typing import Callable, Literal

import numpy as np

import pandas as pd

from sklearn.metrics import accuracy\_score

from .Kernels import \*

from .MethodsOfGetNeighbours import \*

from .Metrics import \*

\_\_all\_\_ = [

'IMetricMethod',

'OneNN',

'KNN',

'ParzenWindowFixedWidth',

'ParzenWindowVariableWidth',

'PotentialFunction',

'KNN\_weight']

class IMetricMethod():

"""Is a generalization for all metric methods"""

\_method : IMethodOfGetNeighbours

\_methods\_factory : MethodOfGetNeighboursFactory

\_metric : IMetric

\_metric\_factory : MetricsFactory

\_y\_train : np.array

def \_\_init\_\_(self, metric : Literal["euclidean", "cityblock", "cosine"],

method : Literal["exhaustive", "kdtree"]) -> None:

"""Common initializer for all metric methods

Args:

metric (str): kind of metric.

method (str): nearest neighbor method.

"""

self.\_metric\_factory = MetricsFactory()

self.\_metric = self.\_metric\_factory.get\_metrics(metric)

self.\_methods\_factory = MethodOfGetNeighboursFactory()

self.\_method = self.\_methods\_factory.get\_method(name\_method=method, metric=metric)

@abstractmethod

def fit(self, data : pd.Series, y\_train : pd.Series) -> None:

"""Data preprocessing

Args:

data (pd.Series): initial data

y\_train (pd.Series): training data class labels

"""

raise NotImplementedError()

@abstractmethod

def \_\_get\_neighbor(self, train\_Y : pd.Series,

nearest\_index : np.ndarray,

distances : np.ndarray) -> any:

"""

Returns the class of the nearest neighbor

Args:

train\_Y (pd.Series): Training sample class labels

data\_point (pd.Series): The object for which the nearest neighbor is found

nearest\_index (np.ndarray): The index of the nearest neighbor

distances (np.ndarray): The distances to the nearest neighbor

Returns:

any: Returns nearest neighbor

"""

raise NotImplementedError()

@abstractmethod

def predict(self, X\_test : pd.DataFrame) -> np.ndarray:

"""

Makes a prediction for X\_test

Args:

X\_test (pd.DataFrame): test set

Returns:

np.ndarray: Array of predicted class labels

"""

raise NotImplementedError()

class OneNN(IMetricMethod):

"""One nearest neighbor"""

def \_\_init\_\_(self, metric : Literal["euclidean", "cityblock", "cosine"] = "euclidean",

method : Literal["exhaustive", "kdtree"] = "exhaustive") -> None:

super().\_\_init\_\_(metric, method)

def \_\_get\_neighbor(self, train\_Y : pd.Series,

nearest\_index : np.ndarray,

distances : np.ndarray = None) -> any:

c\_neighbor = np.take(train\_Y, nearest\_index)

unique, counts = np.unique(c\_neighbor, return\_counts=True)

return unique[np.argmax(counts)]

def fit(self, data : pd.Series, y\_train : pd.Series) -> None:

data = np.array(data)

y\_train = np.array(y\_train)

self.\_method.preprocessing(data)

self.\_y\_train = y\_train

def predict(self, X\_test : pd.DataFrame) -> np.ndarray:

predict = []

for row in np.array(X\_test):

nearest\_index, \_ = self.\_method.get\_neighbours(point=row, knn=1)

predict.append(self.\_\_get\_neighbor(self.\_y\_train, nearest\_index))

return np.array(predict)

class KNN(IMetricMethod):

"""K nearest neighbor"""

\_\_countNeighbor : int

def \_\_init\_\_(self, metric : Literal["euclidean", "cityblock", "cosine"] = "euclidean",

method : Literal["exhaustive", "kdtree"] = "exhaustive",

countNeigbor : int = 10) -> None:

"""

Args:

metric (str): kind of metric. Defaults to "euclidean".

Possible values: "euclidean", "cityblock", "cosine".

method (str): nearest neighbor method. Defaults to "exhaustive".

Possible values: "exhaustive", "kdtree".

countNeigbor (int): number of nearest neighbors. Defaults to 10.

"""

super().\_\_init\_\_(metric, method)

self.\_\_countNeighbor = countNeigbor

def \_\_get\_neighbor(self, train\_Y : pd.Series, nearest\_index : np.ndarray,

distances : np.ndarray = None) -> any:

c\_neighbor = np.take(train\_Y, nearest\_index)

unique, counts = np.unique(c\_neighbor, return\_counts=True)

return unique[np.argmax(counts)]

def fit(self, data : pd.Series, y\_train : pd.Series) -> None:

data = np.array(data)

y\_train = np.array(y\_train)

self.\_method.preprocessing(data)

self.\_y\_train = y\_train

def predict(self, X\_test : pd.DataFrame) -> np.ndarray:

predict = []

for row in np.array(X\_test):

nearest\_index, \_ = self.\_method.get\_neighbours(point=row, knn=self.\_\_countNeighbor)

predict.append(self.\_\_get\_neighbor(self.\_y\_train, nearest\_index))

return np.array(predict)

class KNN\_weight(IMetricMethod):

"""K nearest neighbor"""

\_\_countNeighbor : int

\_\_weight\_func : Callable[[float], float]

def \_\_init\_\_(self, metric : Literal["euclidean", "cityblock", "cosine"] = "euclidean",

method : Literal["exhaustive", "kdtree"] = "exhaustive",

countNeigbor : int = 10,

weight\_func : Callable[[float], float] = lambda x : 1/x) -> None:

"""

Args:

metric (Literal["euclidean", "cityblock", "cosine"]): kind of metric.

weight\_func (Callable[[float], float]): weight function for that

get distance from classifier object and neighbor

method (str): nearest neighbor method. Defaults to "exhaustive".

countNeigbor (int): number of nearest neighbors. Defaults to 10.

"""

super().\_\_init\_\_(metric, method)

self.\_\_countNeighbor = countNeigbor

self.\_\_weight\_func = weight\_func

def \_\_get\_neighbor(self, train\_Y : pd.Series, nearest\_index : np.ndarray,

distances : np.ndarray = None) -> any:

nearest = {cl : 0 for cl in np.unique(train\_Y)}

for ind, dist in zip(nearest\_index, distances):

nearest[train\_Y[ind]] += self.\_\_weight\_func(dist)

return max(nearest, key=nearest.get)

def fit(self, data : pd.Series, y\_train : pd.Series) -> None:

data = np.array(data)

y\_train = np.array(y\_train)

self.\_method.preprocessing(data)

self.\_y\_train = y\_train

def predict(self, X\_test : pd.DataFrame) -> np.ndarray:

predict = []

for row in np.array(X\_test):

nearest\_index, distances = self.\_method.get\_neighbours(point=row, knn=self.\_\_countNeighbor)

predict.append(self.\_\_get\_neighbor(self.\_y\_train, nearest\_index, distances=distances))

return np.array(predict)

class ParzenWindowFixedWidth(IMetricMethod):

"""Variable Width Parzen Window"""

\_\_kernel : IKernel

\_\_kernel\_factory = KernelFactory()

\_\_width : float

def \_\_init\_\_(self, metric : Literal["euclidean", "cityblock", "cosine"] = "euclidean",

method : Literal["exhaustive", "kdtree"] = "exhaustive",

kernel : Literal["rectangular", "gaussian"] = "rectangular",

width : float = 10) -> None:

"""

Args:

metric (str): kind of metric. Defaults to "euclidean".

method (str): nearest neighbor method. Defaults to "exhaustive".

kernel (str): kind of kernel. Defaults to "rectangular".

width (float): window width. Defaults to 10.

"""

super().\_\_init\_\_(metric, method)

self.\_\_kernel = self.\_\_kernel\_factory.get\_kernel(name\_kernel=kernel)

self.\_\_width = width

def \_\_get\_neighbor(self, train\_Y : pd.Series,

nearest\_index : np.ndarray, distances : np.ndarray) -> any:

nearest = {cl : 0 for cl in np.unique(train\_Y)}

for ind, dist in zip(nearest\_index, distances):

nearest[train\_Y[ind]] += self.\_\_kernel.kernel\_func(dist/self.\_\_width)

return max(nearest, key=nearest.get)

def fit(self, data : pd.Series, y\_train : pd.Series) -> None:

data = np.array(data)

y\_train = np.array(y\_train)

self.\_method.preprocessing(data)

self.\_y\_train = np.array(y\_train)

def predict(self, X\_test : pd.DataFrame) -> np.ndarray:

predict = []

for row in np.array(X\_test):

nearest\_index, distances = self.\_method.get\_neighbours(point=row, width=self.\_\_width)

predict.append(self.\_\_get\_neighbor(self.\_y\_train, nearest\_index, distances))

return np.array(predict)

class ParzenWindowVariableWidth(IMetricMethod):

"""Fixed Width Parzen Window"""

\_\_kernel : IKernel

\_\_kernel\_factory = KernelFactory()

\_\_countNeighbor : int

def \_\_init\_\_(self, metric : Literal["euclidean", "cityblock", "cosine"] = "euclidean",

method : Literal["exhaustive", "kdtree"] = "exhaustive",

kernel : Literal["rectangular", "gaussian"] = "rectangular",

countNeighbor : int = 10) -> None:

"""

Args:

metric (str): kind of metric. Defaults to "euclidean".

method (str): nearest neighbor method. Defaults to "exhaustive".

kernel (str): kind of kernel. Defaults to "rectangular".

countNeigbor (int): number of nearest neighbors. Defaults to 10.

"""

super().\_\_init\_\_(metric, method)

self.\_\_kernel = self.\_\_kernel\_factory.get\_kernel(name\_kernel=kernel)

self.\_\_countNeighbor = countNeighbor

def \_\_get\_neighbor(self, train\_Y : pd.Series,

nearest\_index : np.ndarray, distances : np.ndarray) -> any:

width = distances[self.\_\_countNeighbor]

nearest = {cl : 0 for cl in np.unique(train\_Y)}

for ind, dist in zip(nearest\_index[:self.\_\_countNeighbor + 1], distances[:self.\_\_countNeighbor + 1]):

nearest[train\_Y[ind]] += self.\_\_kernel.kernel\_func(dist/width)

return max(nearest, key=nearest.get)

def fit(self, data : pd.Series, y\_train : pd.Series) -> None:

data = np.array(data)

y\_train = np.array(y\_train)

self.\_method.preprocessing(data)

self.\_y\_train = y\_train

def predict(self, X\_test : pd.DataFrame) -> np.ndarray:

predict = []

for row in np.array(X\_test):

nearest\_index, distances = self.\_method.get\_neighbours(point=row, knn=self.\_\_countNeighbor + 1)

predict.append(self.\_\_get\_neighbor(self.\_y\_train, nearest\_index, distances))

return np.array(predict)

class PotentialFunction(IMetricMethod):

"""Method of potential functions"""

\_\_kernel : IKernel

\_\_kernel\_factory = KernelFactory()

\_\_width : int

\_\_eps : float

\_\_potentials : np.ndarray

def \_\_init\_\_(self, metric : Literal["euclidean", "cityblock", "cosine"] = "euclidean",

method : Literal["exhaustive", "kdtree"] = "exhaustive",

kernel : Literal["rectangular", "gaussian"] = "rectangular",

width : float = 0.3, eps : float = 0.05) -> None:

"""

Args:

metric (str): kind of metric. Defaults to "euclidean".

method (str): nearest neighbor method. Defaults to "exhaustive".

kernel (str): kind of kernel. Defaults to "rectangular".

width (float): window width. Defaults to 0.3.

eps (float): error tolerance. Defaults to 0.05.

"""

super().\_\_init\_\_(metric, method)

self.\_\_kernel = self.\_\_kernel\_factory.get\_kernel(name\_kernel=kernel)

self.\_\_width = width

self.\_\_eps = eps

def fit(self, data : pd.Series, y\_train : pd.Series) -> None:

data = np.array(data)

y\_train = np.array(y\_train)

self.\_method.preprocessing(data)

self.\_y\_train = y\_train

self.\_\_potentials = np.zeros(data.shape[0])

# Distance matrix from each vector to each

dist\_matrix = self.\_metric.get\_distance\_matrix(data)

vectors\_dist\_less\_width = {}

# Loop through each vector and check if its distance falls within the distance range

for i, dist in enumerate(dist\_matrix):

vector\_distance = dist

# Get the indexes of vectors that fall within the distance range

included\_indexes = np.where((vector\_distance > 0) & (vector\_distance <= self.\_\_width))[0]

# Add the included indexes to the dictionary with the vector index as the key

vectors\_dist\_less\_width[i] = included\_indexes

err = 1.0

# As long as the number of errors is greater than the specified

while (err > self.\_\_eps):

i = 0

while (i < 1000):

i += 1

# Until we get a class mismatch to update the potentials

rand = np.random.randint(0, len(vectors\_dist\_less\_width))

cl = self.\_\_get\_neighbor(y\_train, vectors\_dist\_less\_width[rand],

dist\_matrix[rand, vectors\_dist\_less\_width[rand]])

if cl != y\_train[rand]:

self.\_\_potentials[rand] += 1

break

# Counting the number of errors

predict = self.predict(data)

err = 1 - accuracy\_score(predict, self.\_y\_train)

def \_\_get\_neighbor(self, train\_Y : pd.Series, nearest\_index : np.ndarray, distances : np.ndarray) -> any:

nearest = {cl : 0 for cl in np.unique(train\_Y)}

for ind, dist in zip(nearest\_index, distances):

nearest[train\_Y[ind]] += self.\_\_kernel.kernel\_func(dist/self.\_\_width) \* self.\_\_potentials[ind]

return max(nearest, key=nearest.get)

def predict(self, X\_test : pd.DataFrame) -> np.ndarray:

predict = []

for row in np.array(X\_test):

nearest\_index, distances = self.\_method.get\_neighbours(point=row, width=self.\_\_width)

predict.append(self.\_\_get\_neighbor(self.\_y\_train, nearest\_index, distances))

return np.array(predict)

**Модуль инициализации пакета (\_\_init\_\_.py)**

from .MetricMethods import \*

1. URL: <https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset> (Дата обращения 04.06.2023) [↑](#footnote-ref-1)
2. URL: https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset (Дата обращения 04.06.2023) [↑](#footnote-ref-2)
3. URL: https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification (Дата обращения 09.06.2023) [↑](#footnote-ref-3)
4. scikit-learn. Официальный сайт [Электронный ресурс]. – Режим доступа: <https://scikit-learn.org/stable/>. [↑](#footnote-ref-4)
5. numpy. Официальный сайт [Электронный ресурс]. – Режим доступа: <https://numpy.org/>. [↑](#footnote-ref-5)
6. pandas. Официальный сайт [Электронный ресурс]. – Режим доступа: <https://pandas.pydata.org/>. [↑](#footnote-ref-6)
7. matplotlib. Официальный сайт [Электронный ресурс]. – Режим доступа: <https://matplotlib.org/>. [↑](#footnote-ref-7)