Обнаружение потенциальных пульсаров

Максим Дергунов

2020

Введение

Пульсары – сильно намагниченные быстро вращающиеся нейтронные звёзды, чьё излучение приходит на Землю через определенные промежутки времени (от 640 импульсов в секунду до 1 импульса в 5 секунд). Они представляют интерес не только для теоретических исследований, но и в практическом плане. Например, российские физики предложили использовать пульсары в качестве эталона времени, ведь их точность превосходит атомный эталон на несколько порядков, а их заряда хватит на несколько миллионов лет. Так же сейчас ведутся разработки систем ориентации по пульсарам для космических спутников, а на золотых пластинах с информацией о человечестве, установленных на космических аппаратах «Пионер» и «Вояджер» местоположение Земли показано относительно 14 ближайших радиопульсаров.

В данном случае мы имеем дело с задачей бинарной классификации. Основная сложность заключается в неравномерном распределении классов – пульсаров всего чуть менее 10% от общего количества объектов в используемом наборе данных. Поскольку данные, пропущенные через модель, пойдут на стол человеку, который будет принимать окончательное решение о присвоении класса «пульсар» звезде, необходимо выбрать модель, которая не только верно классифицирует наибольшее количество звёзд, но и совершает как можно меньше ошибок второго рода, то есть присваивает класс «не-пульсар» пульсару.

Обзор подхода

В данном проекте я постарался найти не только наилучшую модель, но и наилучший способ предобработки данных. Для этого путём обработки оригинального набора данных было создано несколько дополнительных наборов:

- С применением метода главных компонент для уменьшения размерности;
- С применением метода масштабирования;
- С применением трёх способов субдискретизации – случайная субдискретизация, NearMiss-3 и CondensedNearestNeighbour;
- С применением трёх способов передискретизации – случайная передискретизация, SMOTE и ADASYN;

А также комбинации этих методов. На каждом из наборов были обучены восемь алгоритмов машинного обучения - дерево решений, случайный лес, сверхслучайные деревья, логистическая регрессия, метод K-случайных соседей, метод опорных векторов, адаптивный бустинг и градиентный бустинг, - один раз с использованием стандартных гиперпараметров, предлагаемых используемыми библиотеками, и второй раз с поиском по сетке наилучших гиперпараметров.

Набор данных был предварительно разбит на обучающую и тестовую выборки в соотношении 7:3. В результате обучающий набор состоит из 12529 образцов (1129 из них пульсары), а тестовый из 5369 (510 пульсаров).

Целевой метрикой была выбрана Каппа Коэна, но также для каждой модели собраны точность, полнота, f-мера и корреляция Мэтьюса. Обзор используемых методов, алгоритмов и метрик представлен ниже.

Предобработка

Метод главных компонент

Метод главных компонент (Principal Component Analysis) – один из основных методов уменьшить размерность данных, потеряв при этом наименьшее количество информации. Суть метода в ортогональном линейном преобразовании, которое отображает данные из исходного пространства признаков в новое пространство меньшей размерности. Цель его применения – ценой небольшой потери качества сильно ускорить работу модели.

На рисунке 1 показано снижение размерности исходного двумерного пространства до одномерного. Видно, что первая главная компонента PC_1 ориентирована вдоль наибольшей вытянутости эллипсоида исходных данных. При этом проекция дисперсии данных на ось главной компоненты D_{pc1} больше, чем её проекции на исходные оси D_{x1} и D_{x2}, но меньше их суммы, то есть с помощью одной главной компоненты выразить всю дисперсию данных не удалось. Поэтому строят вторую, третью и так далее главные компоненты, пока вся дисперсия не будет выражена.

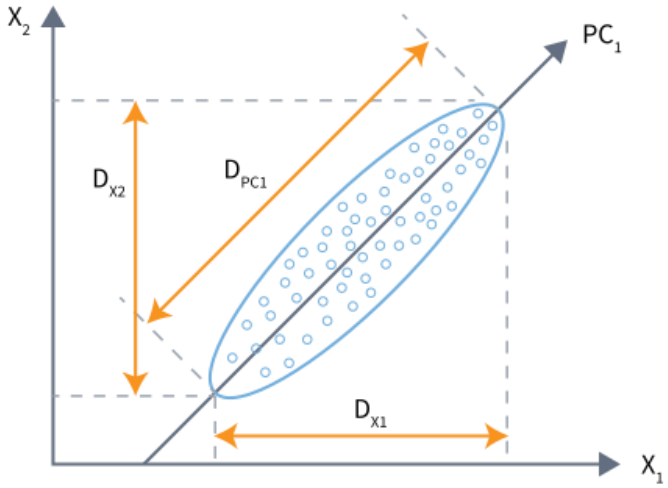

Рисунок 1. Метод главных компонент

При этом вдоль каких-то осей дисперсия может быть больше или меньше. Предполагается, что чем меньшая дисперсия вдоль конкретной оси, тем меньший вклад она вносит. В результате мы отбрасываем главные компоненты, дисперсия вдоль которых минимальна, и оставляем заданное количество главных компонент с максимальной дисперсией.

Стандартизация

Стандартизация – это процедура, в ходе которой значения признаков масштабируются таким образом, что приобретают свойства стандартного нормального распределения с $\mu = 0$ и $\sigma = 1$, где μ – среднее значение признака, а σ – стандартное отклонение от среднего.

Поскольку данные в нашем наборе распределены неравномерно и могут принимать достаточно большие значения, был использован метод устойчивого масштабирования (Robust Scaler). Он работает следующим образом:

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}.$$

В отличие от стандартного масштабирования, в устойчивом вместо среднего значения и стандартного отклонения используются квартили. Квартили – это числа, которые делят значения

признаков в наборе данных на четыре равные части. Нижний квартиль Q_1 – это число, которое больше или равно четверти наименьших значений признака. Соответственно, верхний квартиль Q_3 – число, которое меньше или равно четверти наибольших значений признака.

Используя квартили возможно игнорировать образцы, которые сильно отличаются от остальных.

Дискретизация данных

Классы в наборе данных распределены неравномерно – 16259 отрицательных и 1639 положительных образцов. Такое распределение представляет собой проблему, поскольку бóльшая часть алгоритмов машинного обучения сильно теряют в качестве при неравномерном распределении классов. Для решения этой проблемы были использованы методы субдискретизации, когда мы удаляем некоторое количество образцов большинства, и передискретизации, когда мы генерируем новые образцы меньшинства.

Для иллюстрации результатов работы с помощью `sklearn.datasets.make_classification` был сгенерирован набор данных. Были использованы такие параметры генератора: *n_samples=10000, n_features=2, n_informative=2, weights=[0.1,0.9], class_sep=1.2.*

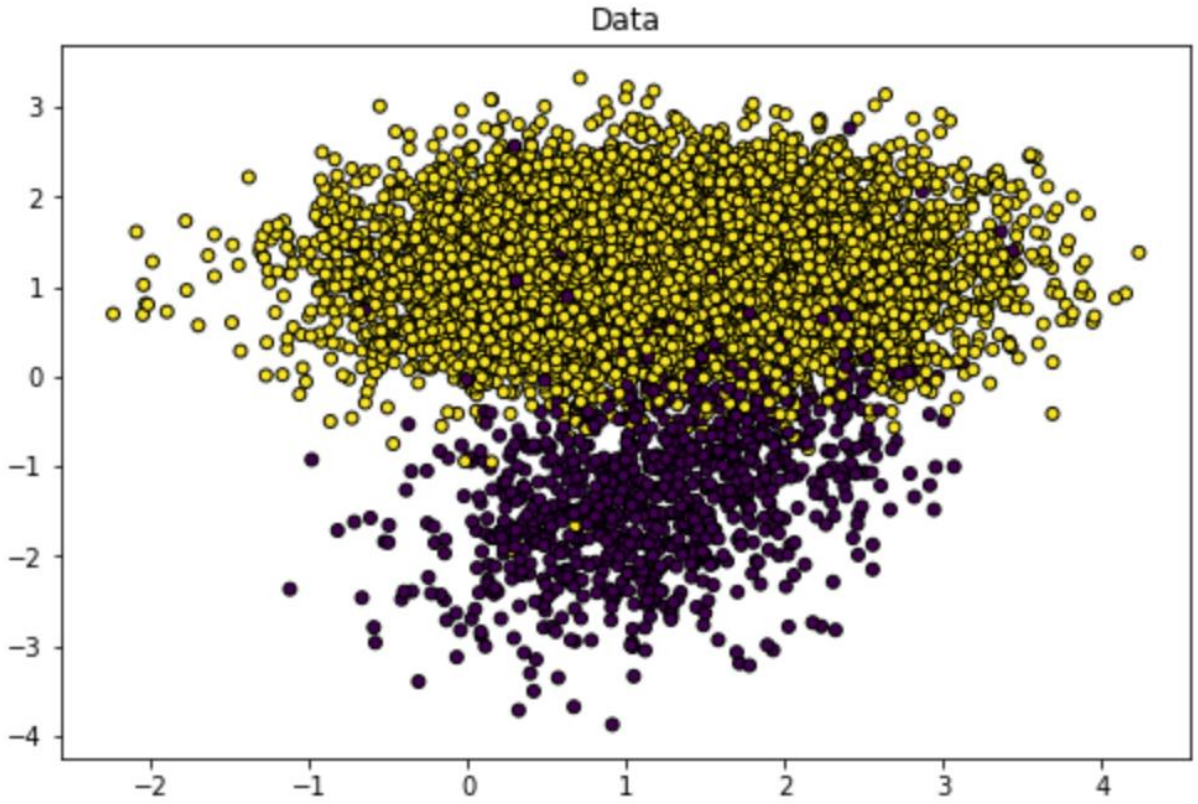

Рисунок 2. Синтетический набор данных

Субдискретизация

Случайная субдискретизация (Random Undersampling)

При использовании метода случайной субдискретизации удаляются случайные образцы большинства до какого-то соотношения. В данной работе количество классов уравнивалось.

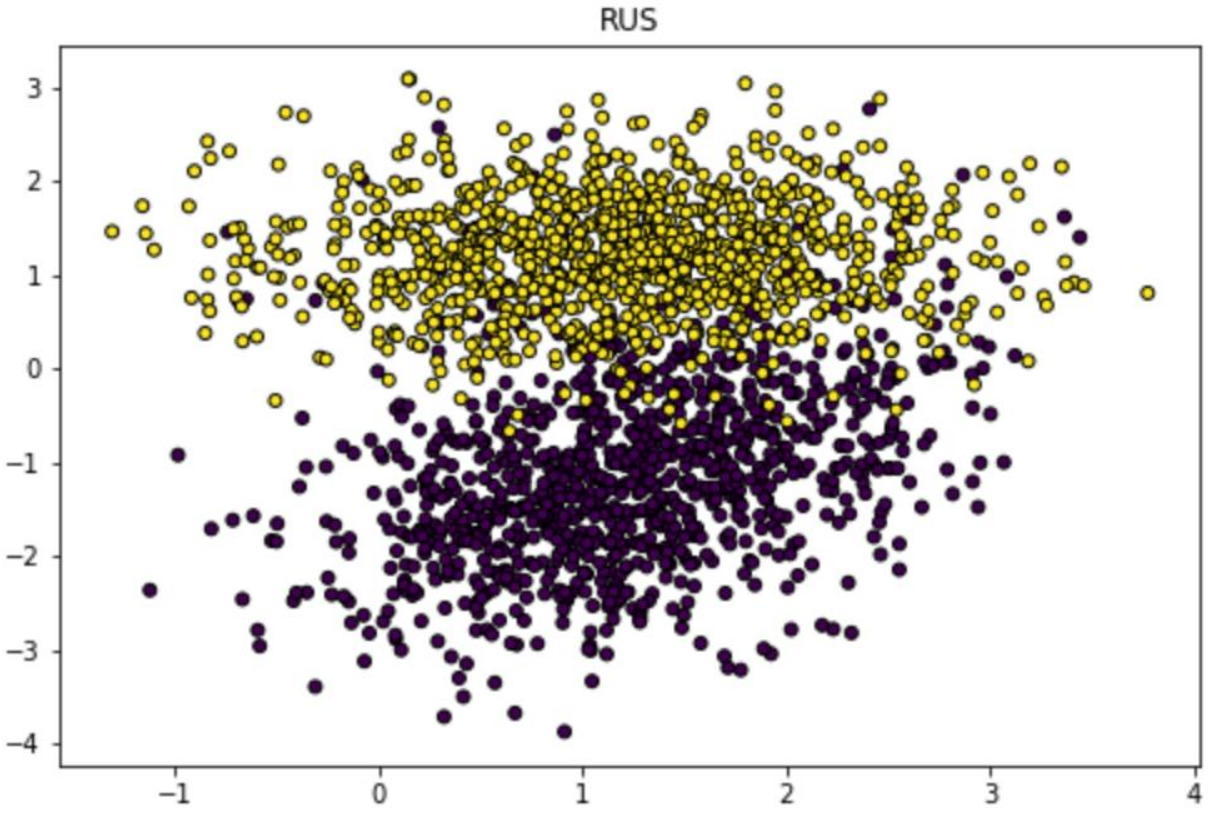

Рисунок 3. Результат работы алгоритма случайной передискретизации

Near Miss

Существуют три версии данного алгоритма. В первой сохраняются такие образцы большинства, чьё среднее евклидово расстояние до *k* образцов миноритарного класса наименьшее. В данном проекте использовалось стандартное значение *k = 3.* Вторая версия наоборот выбирает образцы с максимальным средним расстоянием. Алгоритмы машинного обучения, работавшие с наборами данных, обработанными с помощью этих двух версий, показали значительно худшие результаты в сравнении с третьей версией, поэтому в проекте использован только алгоритм Near Miss версии 3.

Эта версия работает следующим образом: сначала для каждого образца меньшинства выбирается *M* ближайших соседей из числа образцов большинства. Затем из них выбираются образцы с максимальным расстоянием до *N* ближайших образцов меньшинства. Значения M и N в проекте были равны 3. В результате в получившихся наборах данных образцов большинства оказалось меньше, чем меньшинства!

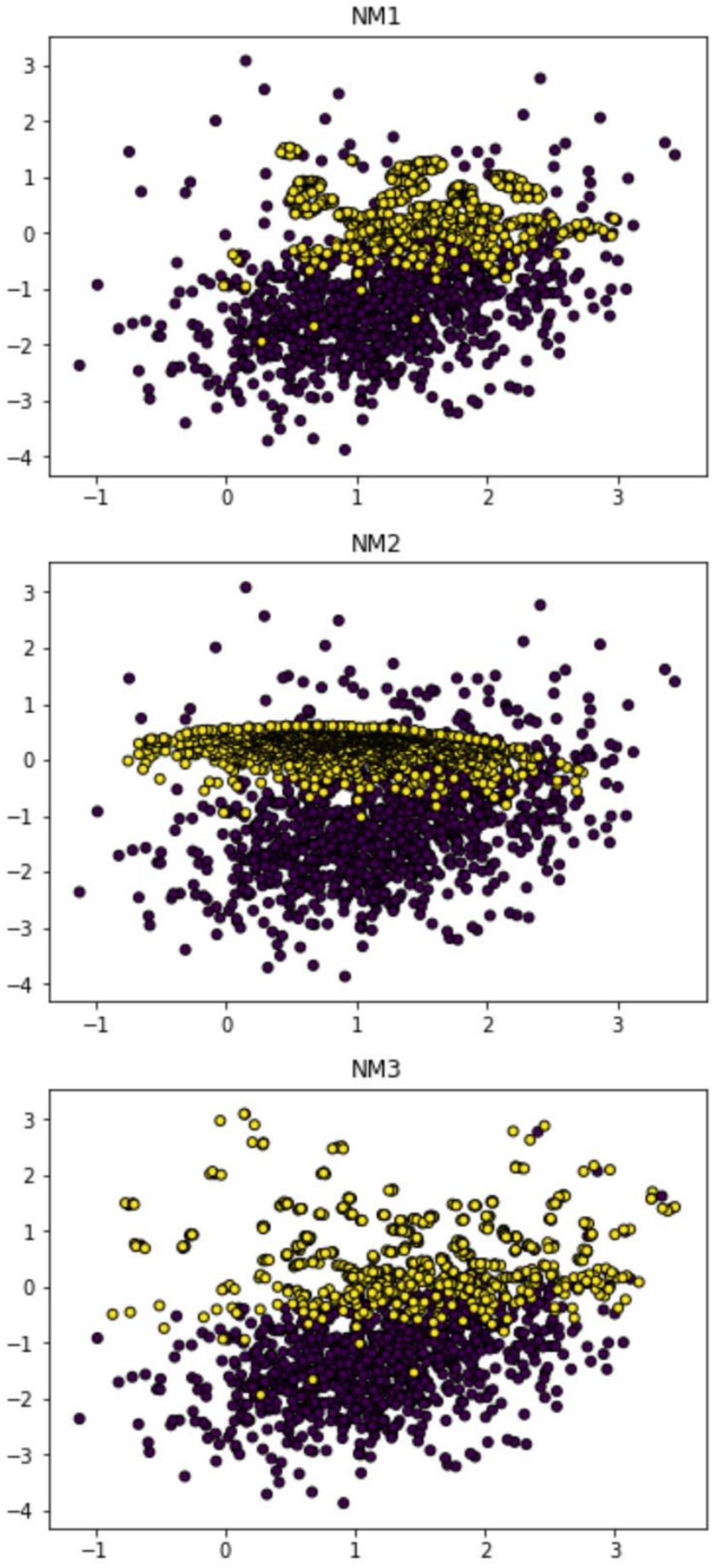

Рисунок 4. Результат работы трёх версий алгоритма NearMiss

Метод уплотнённых ближайших соседей (Condensed Nearest Neighbors)

Основная идея в том, чтобы удалить образцы большинства вокруг случайного образца. Работает это следующим образом: имея набор данных *S,* создаётся новый набор *C*, куда входят все образцы меньшинства и один случайно выбранный образец большинства. Затем мы классифицируем набор S методом *k* случайных соседей с *k* = 1 используя точки из C. Все образцы, которые классифицированы неверно, переносятся из *S* в *C*. Алгоритм повторяется до тех пор, пока не остаётся нечего переносить.

Основным недостатком данного метода является его низкая скорость из-за необходимости проходить через набор данных снова и снова. Так же из-за случайности выбора первого образца для переноса в набор *C* результаты могут меняться.

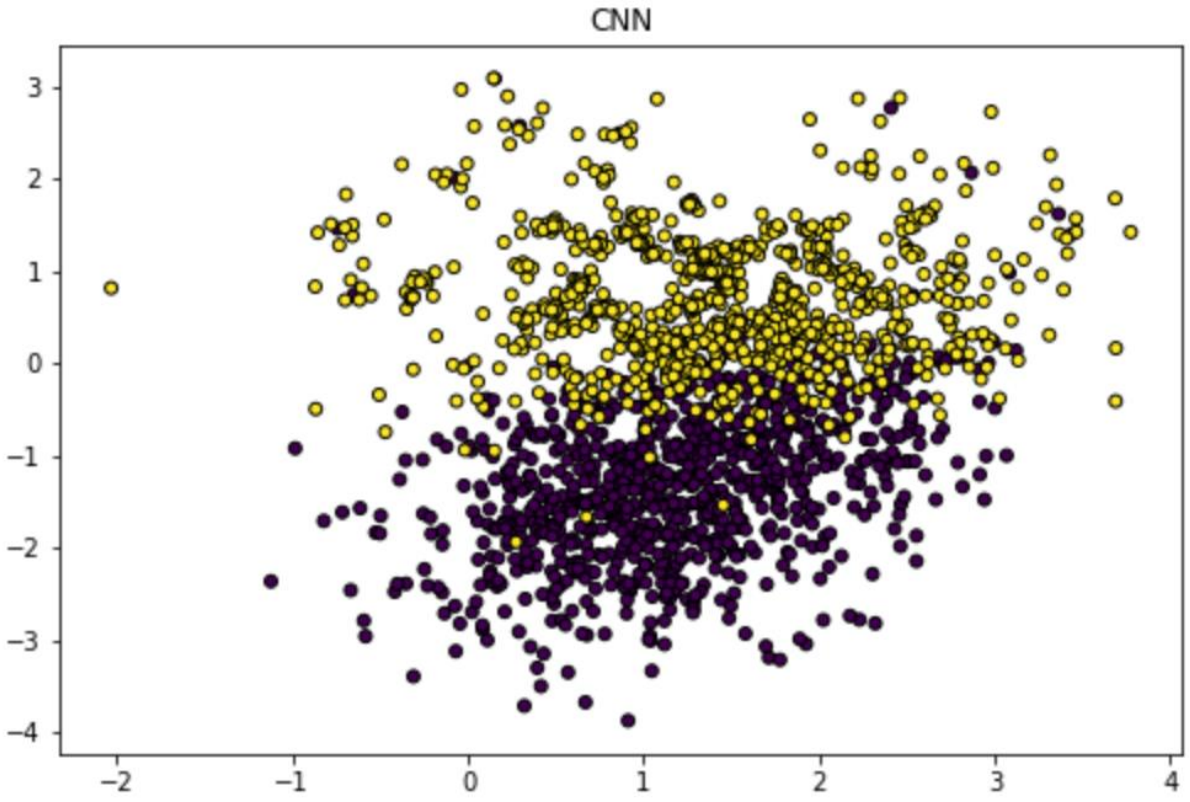

Рисунок 5. Результат работы метода уплотнённых ближайших соседей

Передискретизация

Случайная передискретизация (Random Oversampling)

Случайная передискретизация подразумевает простое дублирование образцов меньшинства. Данный метод зачастую приводит к переобучению.

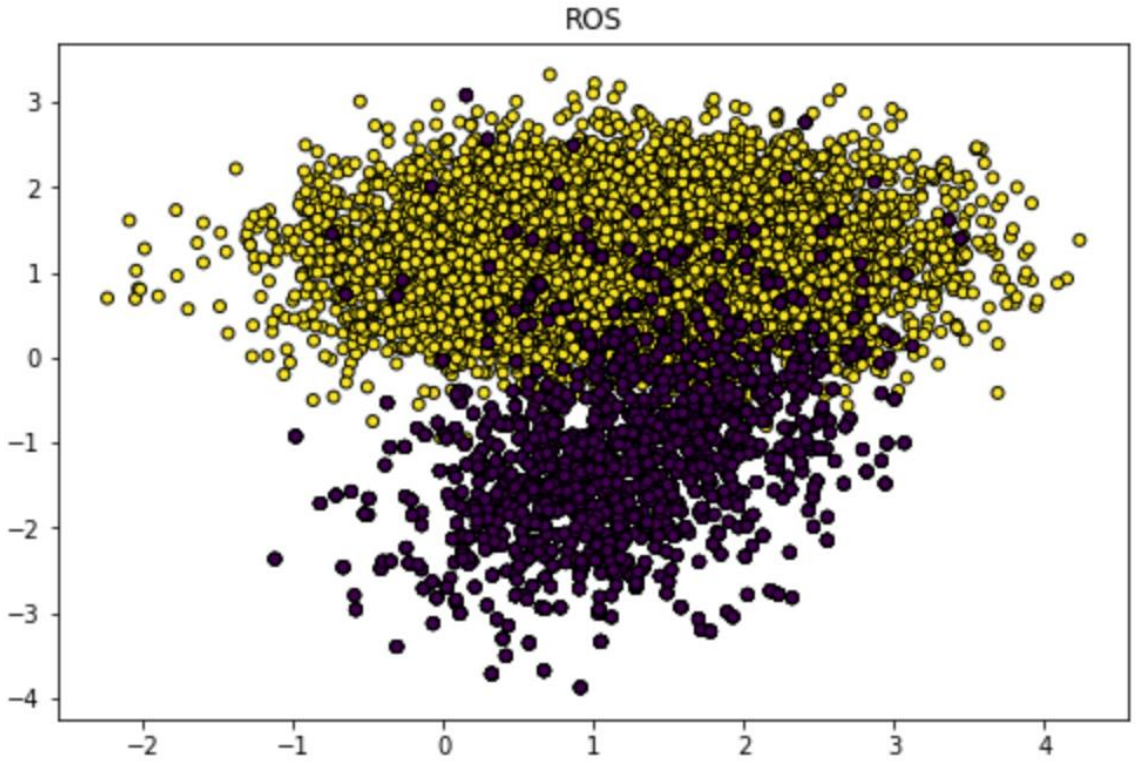

Рисунок 6. Результат работы алгоритма случайной передискретизации

SMOTE (Synthetic Minority Oversampling TEchnique)

В отличие от предыдущего метода в этом генерация новых образцов производится уже на основании реальных данных: выбираются случайные образцы, находятся их ближайшие соседи и новая точка генерируется между образцом и его соседом.

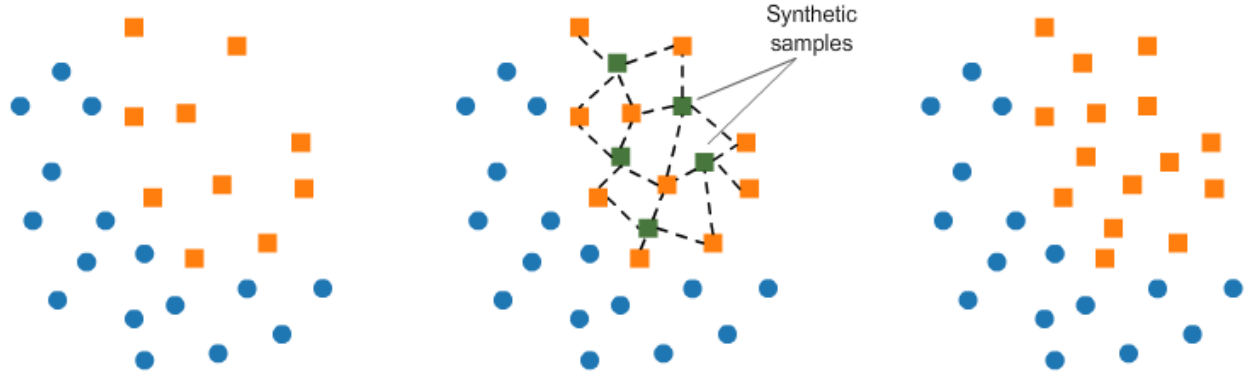

Рисунок 7. Иллюстрация работы алгоритма SMOTE

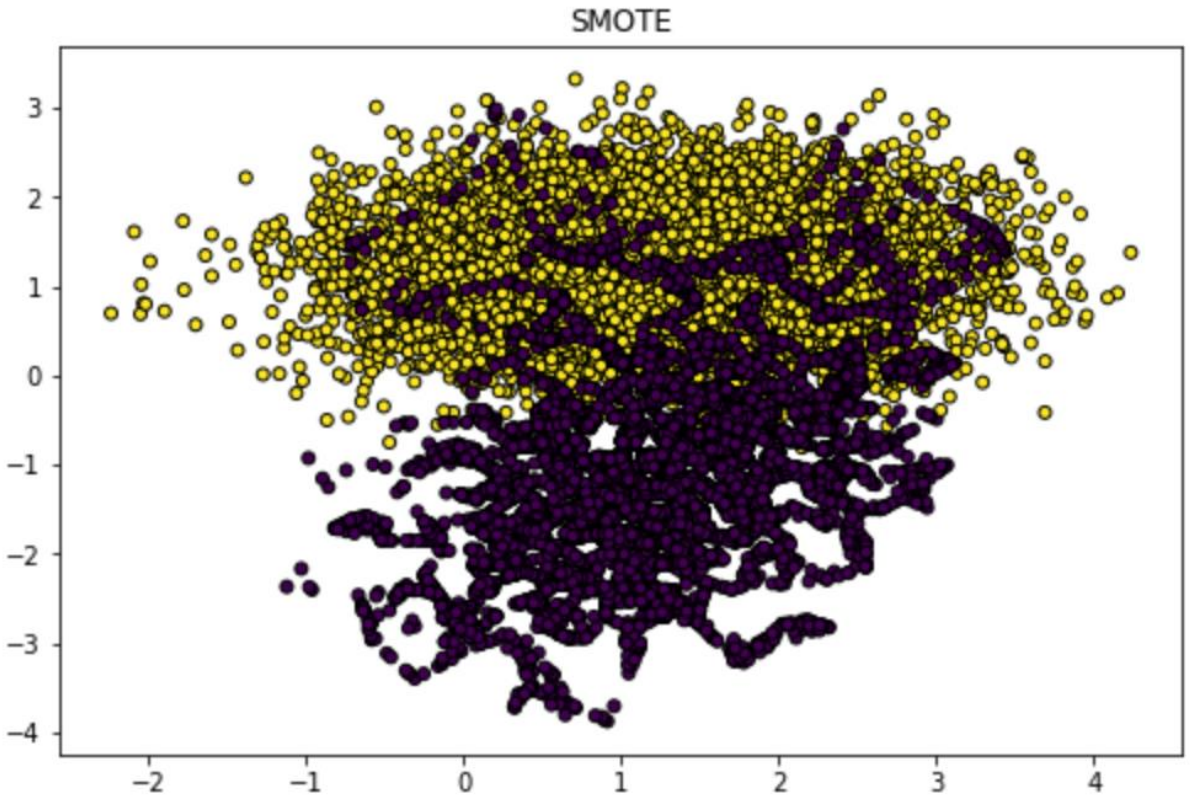

Рисунок 8. Результат работы алгоритма SMOTE

ADASYN (Adaptive Synthetic)

Наконец, метод ADASYN основывается на SMOTE – здесь тоже вычисляются ближайшие соседи, но учитывается ещё и распределение плотности, то есть соотношение классов меньшинства и большинства. Для тех образцов меньшинства, которое сложнее классифицировать, генерируется больше синтетических данных. Повторяя этот процесс, алгоритм сдвигает границу решения.

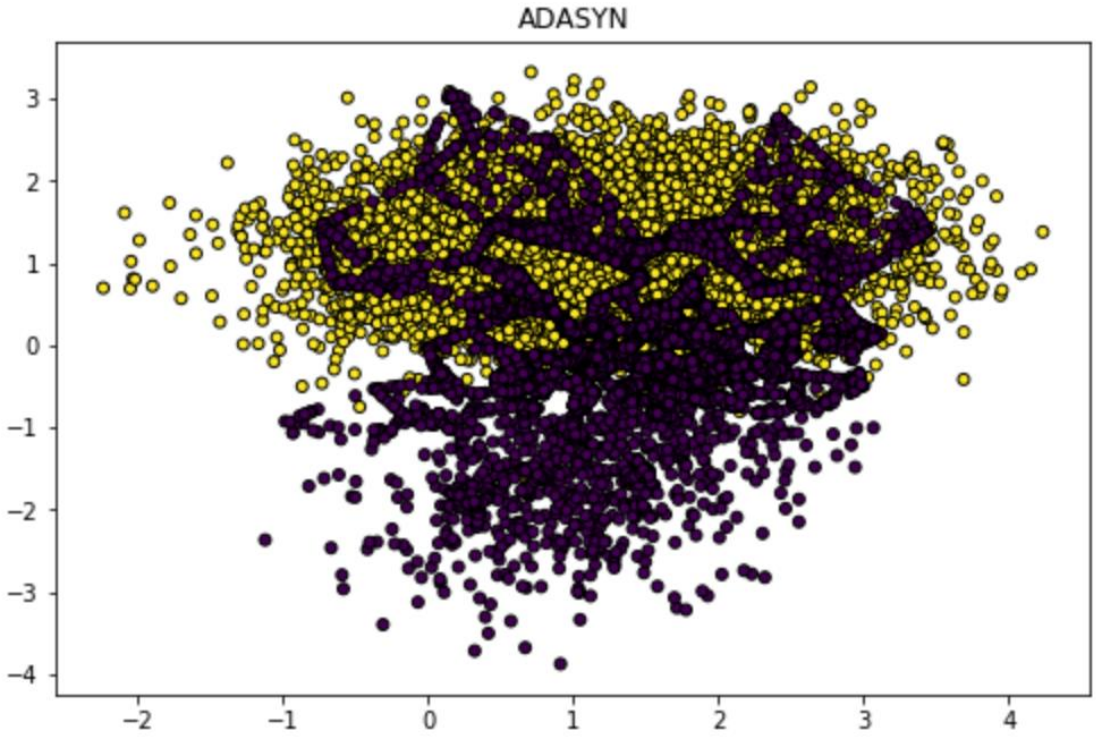

Рисунок 9. Результат работы алгоритма ADASYN

Таблица 1. Итоговое распределение классов

Алгоритмы обучения

Метод k-ближайших соседей

Это один из самых простых алгоритмов. Он относит объект к классу, к которому относятся k его ближайших соседей в многомерном пространстве. Например, при k=3 каждый объект сравнивается с тремя ближайшими соседями. Геометрическое расстояние между двумя точками в многомерном пространстве, также называемое Евклидовым расстоянием, рассчитывается по формуле:

$$d_{xy} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}.$$

Метод опорных векторов (Support Vector Machine)

Метод опорных векторов рассматривает каждый вектор признаков как точку в многомерном пространстве (в нашем случае это восьмимерное пространство). Алгоритм размещает эти точки на воображаемом графике и рисует гиперплоскость, разделяющую данные с положительным классом от данных с отрицательным классом. Гиперплоскость задаётся уравнением с двумя параметрами: вектором w той же размерности, что и входной вектор признаков x, и действительным числом b:

$$wx - b = 0,$$

где wx означает $w_{(1)}x_{(1)} + w_{(2)}x_{(2)} + \ldots + w_{(D)}x_{(D)}$, а D – число измерений в векторе x.

Теперь можно спрогнозировать класс таким образом:

$$y = sign(wx - b),$$

где $sign$ это математический оператор, принимающий какое-то значение и возвращающий 1, если входное значение является положительным числом, и -1, если входное значение отрицательно.

Таким образом, цель алгоритма в том, чтобы найти оптимальные значения параметров w и b.

Логистическая регрессия

Несмотря на название, метод логистической регрессии это всё же метод классификации. Название пришло из статистки и объясняется тем, что математическая формулировка метода аналогична линейной регрессии.

Этот метод позволяет узнать вероятность принадлежности образца к какому-то классу. Для этого необходимо получить не только определенную линейную комбинацию признаков $wx_i + b$, но и подставить эту комбинацию в непрерывную функцию с областью значений (0,1). Одной из таких функций является стандартная логическая функция:

$$f(x) = \frac{1}{1+e^{-x}},$$

где e – основание натурального алгоритма.

Таким образом модель логистической регрессии выглядит так:

$$f(x) = \frac{1}{1+e^{-(wx+b)}}.$$

Подобрав параметры и применив модель мы получим значение $0 < p < 1$. В результате мы получаем вероятность того, что какой-то образец будет отнесён к положительному классу и если это значение больше определенного порога (по умолчанию он равен 0.5), то образцу присваивается положительный класс.

Дерево решений

Проще всего проиллюстрировать метод построения дерева решений на примере игры в 20 вопросов. Суть игры такова: ведущий загадывает какого-то человека, чаще всего знаменитость, а участник должен отгадать кто это, задавая вопросы на да и нет. Какой вопрос задать первым? Вопрос «это Карл Саган?» удалит из множества возможных ответов только один вариант, тогда как вопрос о поле загаданного человека сразу уменьшит это множество примерно в два раза. Другими словами, признак «Пол» разделяет набор данных о знаменитостях лучше, чем признак «спортсмен» или «выращивает бонсаи». Таким образом мы подходим к концепции получения информации на основе энтропии.

Энтропия Шеннона для системы с N возможных состояний определяется как:

$$S = -\sum_{i=1}^{N} p_i log_2 p_i,$$

где p – функция вероятности. Чем выше энтропия, тем более хаотична система. Измеряя энтропию в получившихся после разделения наборах, мы можем оценить эффективность этого разделения:

$$IG(Q) = S_0 - \sum_{i=1}^{q} \frac{Ni}{N} S_i,$$

где Q – условие, на основе которого происходит разбиение, q – количество получившихся наборов, N_i – количество соответствующих условию Q объектов в новом наборе.

Алгоритм ищет наиболее эффективные варианты для разделения и делит набор до тех пор, пока не останется нечего делить (что, как правило, приводит к переобучению), либо пока не будет достигнут задаваемый аналитиком максимальный предел глубины дерева.

Случайный лес

Случайный лес – один из немногих универсальных алгоритмов. Он может использоваться для решения задач не только классификации, но и регрессии, кластеризации, поиска аномалий и т.д.

Если мы хотим построить лес из N деревьев необходимо:

- Для каждого $n = 1, ..., N$:
    - Создать выборку X_n из обучающего набора;
    - Построить дерево b_n на выборке X_n;
        - Для каждого расщепления по заданному критерию выбрать лучший признак и так до исчерпания выборки;
        - Дерево строится, пока не достигнет заданной высоты или пока в каждом листе не окажется заданное количество объектов;
        - При каждом разбиении выбираются заданные m признаков из n исходных и оптимальное расщепление ищется среди них.

В результате для задач классификации итоговый результат выбирается голосованием по большинству.

Сверхслучайные деревья

Основное отличие метода сверхслучайных деревьев от случайного леса состоит в расщеплении узла. Так же, как и в случайном лесу, используется случайное подмножество признаков, но вместо поиска наилучшего порога, для каждого возможного признака выбирается случайный порог. Затем каждый порог оценивается в соответствии с выбранным критерием и на основании лучшего производится расщепление.

Адаптивный бустинг (AdaBoost)

В этом алгоритме строятся не полноценные деревья, как в случайном лесу, а так называемые «пеньки», то есть деревья с глубиной равной единице. По отдельности каждый такой пень ужасно классифицирует образцы, но это компенсируется тем, что в итоговом лесу вес их голосов не равноценен. Кроме того, если в обычном случайном лесу порядок создания каждого отдельного леса неважен, то в AdaBoost каждый новый пень создаётся с учётом ошибок предыдущих.

Вначале каждому образцу присваивается одинаковый вес $\frac{1}{\text{общее число образцов}}$. Затем создаётся первый пень, который ищет наилучшее расщепление по каждому признаку. Оценка каждого расщепления происходит по значению индекса Джини:

$$Gini = 1 - \sum_k (p_k)^2,$$

где k – количество классов, а p_k – вероятность присваивания класса. Выбирается пень, у которого значение индекса наименьшее.

Затем необходимо рассчитать вес его голоса. Для этого необходимо рассчитать общую ошибку – суммарный вес всех неправильно классифицированных образцов. Рассчитав ошибку, можно получить значение веса пня:

$$\text{вес пня} = \frac{1}{2}\log\left(\frac{1 - \text{общая ошибка}}{\text{общая ошибка}}\right)$$

Таким образом получается, что если общая ошибка мала, то вес пня будет большим положительным числом. Если же ошибка велика, то вес будет отрицательным. Другими словами, если пень указывает, что образец необходимо причислить к пульсарам, то на этапе голосования отрицательный вес превратит это в голос за причисление к не-пульсарам.

Теперь необходимо перераспределить веса образцов. Начнём с некорректно классифицированных образцов – их новые веса рассчитываются по формуле

$$\text{новый вес} = \text{старый вес} \times e^{\text{вес пня}}$$

Для верно классифицированных образцов вес необходимо понизить:

$$\text{новый вес} = \text{старый вес} \times e^{-\text{вес пня}}$$

Новые веса нормализуются (каждый вес делится на сумму весов). Теперь создаётся новый набор данных, по размерности равный предыдущему. В этот новый набор выбираются образцы из предыдущего, при этом у образцов с большими весами шанс попасть в новый набор выше, к тому же они могут дублироваться. Затем образцам в новом наборе присваиваются одинаковые веса. Несмотря на это, за счёт того, что в новом наборе есть дублирующие образцы, ошибка в их классификации будет стоить дороже. Алгоритм повторяется, пока наборы данных не перестанут меняться или пока наберётся указанное количество пней.

Теперь, когда лес пней готов, происходит голосование. Каждый пень делает прогноз и считается суммарный вес пней, которые относят образец к пульсарам и тех, что относят его к не-пульсарам. Итоговое решение принимается на основании стороны, набравшей наибольший вес.

Градиентный бустинг

В работе использовалась реализация метода градиентного бустинга из библиотеки XGBoost. Цель леса в данном случае – получение вероятности того, что образец будет отнесен к целевому классу.

Вначале задаётся начальное предсказание. По умолчанию оно равно 0.5. Затем для каждого образца рассчитывается остаток – разность между наблюдением и начальным предсказанием. То

есть, на начальном этапе остаток для любого образца будет равен 0.5 или -0.5. Строится решающее дерево, которое будет предсказывать эти остатки. Но мы не можем просто использовать эти предсказания – нам нужно трансформировать их:

$$\frac{(\sum(\text{остаток}_i)^2}{\sum \text{предыдущее предсказание}_i \times(1-\text{ предыдущее предсказание}_i)+\lambda},$$

где λ – параметр регуляризации. Мы производим этот расчёт для каждого листа дерева.

Теперь, когда дерево готово, необходимо получить интересующие нас вероятности. Сначала необходимо преобразовать наше изначальное предсказание в логарифмическую форму по формуле $\log(\frac{p}{1-p})$ – для стандартного значения 0.5 мы получим 0. Теперь мы складываем результат с произведением результата дерева и коэффициента скорости обучения. Получившееся число мы передаём в сигмоидную функцию и получаем вероятность принадлежности объекта к классу.

Получив новые вероятности, можно получить новые остатки и начинать строить новое дерево.

Алгоритм повторяется до тех пор, пока вероятности не перестанут изменяться либо пока не будет достигнуто заданное количество деревьев. Наконец, каждое дерево выдаёт свой прогноз, каждый прогноз умножается на коэффициент скорости обучения и они суммируются вместе с логарифмической формой изначального предсказания. Результат обрабатывается сигмоидной функцией и мы получаем вероятность.

Выбор метрики

Самая очевидная и легкая для интерпретации метрика качества классификации – точность (accuracy) – не подходит для решаемой задачи из-за неравномерного распределения классов. Поэтому целевой метрикой в работе была выбрана Каппа Коэна, а также собраны значения точности (precision; не путать с accuracy), полноты (recall), f1-меры и корреляции Мэтьюса.

Вначале необходимо понять, что такое матрица ошибок. В задаче бинарной классификации может выдавать не только Истинно Положительные (True Positive, TP) и Истинно Ложные (True Negative, TN), но и Ложно Положительные (False Positives, FP) и Ложно Отрицательные (False Negative, FN). Тогда матрица ошибок будет выглядеть следующим образом:

	$y = 1$	y = 0
$\hat{y} = 1$	*True Positive*	*False Positive*
$\hat{y} = 0$	*False Negative*	*True Negative*

Здесь $\hat{y}$ – ответ алгоритма на объекте, а y – истинная метка класса объекта.

Точность (precision)

$$Precision = \frac{TP}{TP + FP}$$

Эту метрику можно интерпретировать как долю объектов, которые классификатор назвал положительными и которые при этом действительно являются положительными.

Полнота (recall)

$$Recall = \frac{TP}{TP + FN}$$

Полнота показывает какую долю объектов положительного класса обнаружил алгоритм.

F-мера (F1-score)

$$F = 2\frac{Precision \times Recall}{Precision + Recall}$$

F-мера представляет собой метрику, которая объединяет в себе информацию о точности и полноте классификатора. Фактически это гармоническое среднее между ними.

Коэффициент корреляции Мэтьюса (Matthews Correlation Coefficient)

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Этот коэффициент учитывает все значения матрицы ошибок. Он принимает значения из интервала от -1 до +1. Модель, получившая оценку +1 является идеальной, 0 – выдаёт случайные предсказания, а -1 – обратные предсказания.

Каппа Коена (Cohen's Kappa)

Каппа Коэна оценивает согласованность между наблюдением и предсказанием. Вначале необходимо рассчитать наблюдаемую согласованность. Для бинарной классификации она рассчитывается так:

$$P_0 = \frac{TP + TN}{TP + TN + FP + FN}$$

Теперь рассчитываем случайную согласованность:

$$P_e = \frac{TP + FP}{TP + TN + FP + FN} \times \frac{TP + FN}{TP + TN + FP + FN} + \frac{FN + TN}{TP + TN + FP + FN} \times \frac{FP + TN}{TP + TN + FP + FN}$$

И, наконец, рассчитывается сама каппа Коэна:

$$k = \frac{P_0 - P_e}{1 - P_e}$$

Как и коэффициент корреляции Мэтьюса, каппа принимает значения от -1 до +1.

Подбор гиперпараметров

Для подбора гиперпараметров использован алгоритм поиска по сетке. Вначале для каждого алгоритма задаются гиперпараметры и их значения (или диапазон значений). Обучающая выборка разбивается на пять фолдов, причём разбивка стратифицирована, то есть в каждом фолде распределение классов соответствует распределению в полной тестовой выборке. Наконец, для каждой комбинации гиперпараметров строится пять моделей – вначале первый фолд используется для валидации, а остальные для обучения, потом второй, и т.д. Результаты оцениваются выбранной метрикой. Пять результатов усредняются и по этому числу происходит оценка модели. Конечная цель – найти самую лучшую комбинацию гиперпараметров для данного алгоритма и данных.

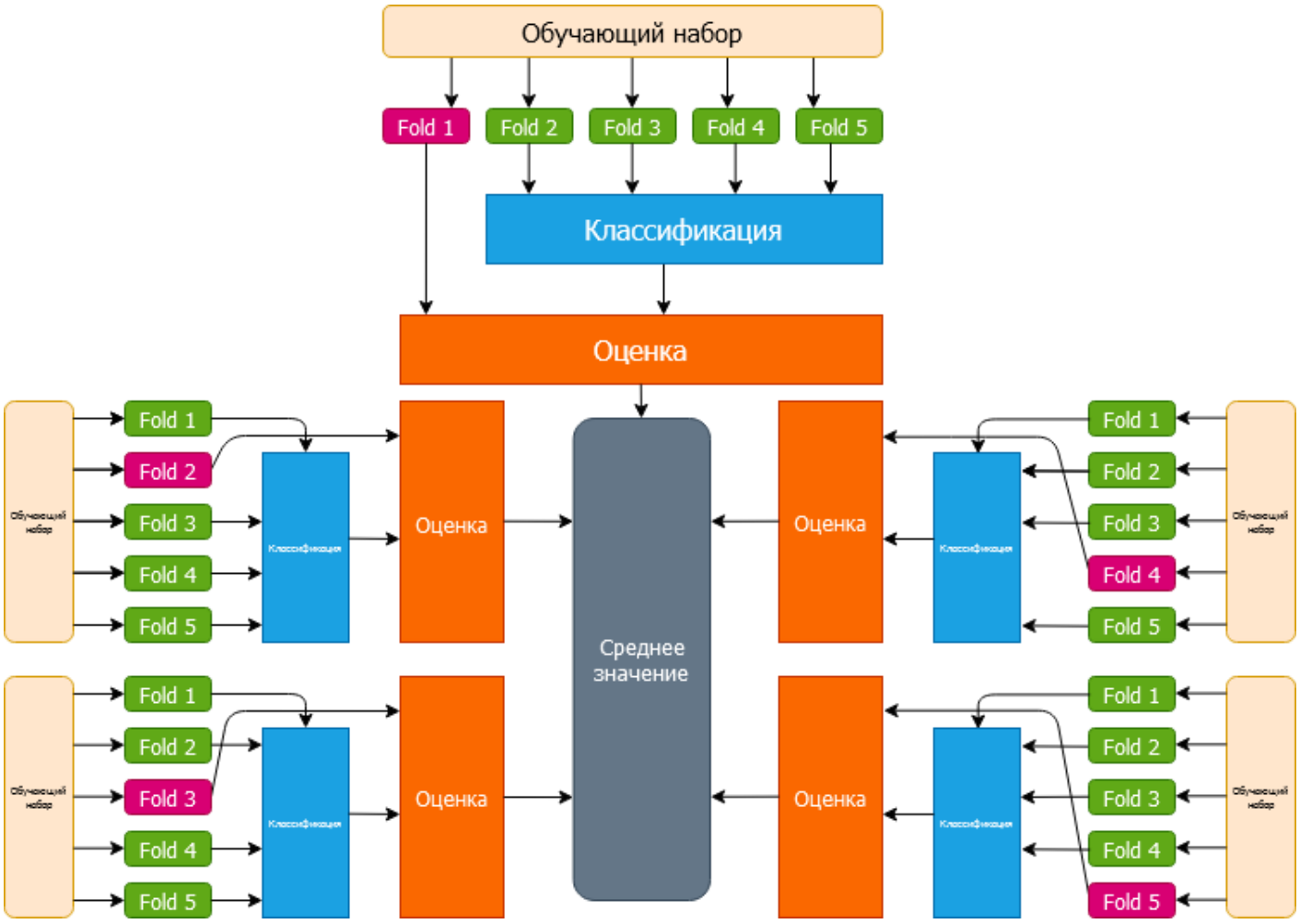

Рисунок 10. Кросс-валидация с 5 фолдами

Таблица 2. Параметры, среди которых проводился поиск

Метод опорных векторов	*C = np.geomspace(0.8, 1.2, 5)* *kernel = ['linear', 'poly', 'rbf']* *gamma = ['scale', 'auto']*
Логистическая регрессия	*penalty = ['l1', 'l2']* *C = np.linspace(0.1, 10, 50),* *solver = ['liblinear', 'newton-cg', 'lbfgs']* *max_iter = [50, 100, 500]*
Дерево решений	*splitter = 'best', 'random'* *max_depth = range(3, 9)* *max_features = ['auto', 'log2', None]*
Случайный лес	*n_estimators = [400, 200, 100]* *criterion = ['entropy']* *max_depth = [10, 9, 8, 7]* *max_features = ['auto', 'log2']*
Сверхслучайные деревья	*n_estimators = [50, 100, 500, 1000]* *criterion = ['entropy']* *max_depth = range(5, 9)* *max_features = ['auto', 'log2', None]*
Адаптивный бустинг (AdaBoost)	*n_estimators = range(500, 1100, 100)* *algorithm = ['SAMME', 'SAMME.R']* *learning_rate = np.linspace(0.01, 0.3, 5)*
Градиентный бустинг	*max_depth = [3, 4]* *learning_rate = [0.3]* *booster = ['dart']* *subsample = [0.7]* *colsample_bylevel = [0.7, 0.6],* *colsample_bynode = [0.7, 0.6]* *colsample_bytree = [0.7, 0.6]* *gamma = [0]* *n_estimators = [1000, 100, 50]* *tree_method = [‘exact’]* *‘rate_drop’ = [0.03]*
k-случайных соседей	*n_neighbors = range(3, 15)* *weights ='uniform', 'distance'* *algorithm= 'ball_tree', 'kd_tree', 'brute'*

Результаты

Всего было построено 312 моделей – по 48 вариантов для каждого алгоритма, за исключением градиентного бустинга и метода опорных векторов. В связи с их долгим временем обучения было решено исключить варианты с применением метода главных компонент и без скалирования. Причина проста – модели, в которых применялся этот метод или отсутствовало скалирование показали себя значительно хуже при работе других алгоритмов. Так, лучшая модель с применением метода главных компонент заняла 66 место, а модель без скалирования – 103.

Ниже представлены таблица с параметрами и значениями метрик десяти лучших моделей, а также матрицы ошибок этих моделей.

Таблица 3. Итоговые результаты

Алгоритм	Метод главных компонент	Подбор гиперпараметров	Скалирование	Сэмплирование	Взвешенная точность	Взвешенная полнота	Взвешенная F-мера	Параметры модели	Коэффициент корреляции Мэтьюса	Каппа Коэна
Градиентный бустинг	False	True	True	SMOTE	0.980401	0.980630	0.980491	*max_depth = 4 learning_rate = 0.3 booster = 'dart' subsample = 0.7 colsample_bylevel = 0.7 colsample_bynode = 0.7 colsample_bytree = 0.7 gamma = 0 n_estimators = 1000 tree_method = 'exact' rate_drop = 0.03*	0.886	0.886
Случайный лес	False	True	True	SMOTE	0.980073	0.980257	0.980152	*n_estimators = 400 criterion = 'entropy' max_depth = 10 max_features = 'log2'*	0.884	0.884
Логистическая регрессия	False	True	False	Случайная передискретизация	0.979958	0.980257	0.980062	*penalty = 'l2' C = 0.98 solver = 'newton-cg' max_iter = 50*	0.883	0.883
Логистическая регрессия	False	True	True	Случайная передискретизация	0.979939	0.980257	0.980043	*penalty = 'l2' C = 3.74 solver = 'lbfgs' max_iter = 50*	0.883	0.883
Логистическая регрессия	False	False	True	Случайная передискретизация	0.979758	0.980071	0.979864	*penalty = 'l2' C = 1 solver = 'lbfgs max_iter = 100*	0.882	0.882
Логистическая регрессия	False	False	True	SMOTE	0.979758	0.980071	0.979864	*penalty = 'l2' C = 1 solver = 'lbfgs' max_iter = 100*	0.882	0.882
Случайный лес	False	True	True	NM-3	0.979758	0.980071	0.979864	*n_estimators = 200 criterion = 'entropy' max_depth = 8 max_features = 'auto'*	0.881	0.882
Логистическая регрессия	False	True	True	SMOTE	0.979671	0.980071	0.979752	*penalty = 'l1' C = 0.7 solver = 'liblinear' max_iter = 50*	0.881	0.880
Адапативный бустинг	False	True	True	SMOTE	0.979444	0.979698	0.979544	*n_estimators = 1000 algorithm = 'SAMME.R' learning_rate = 0.08*	0.881	0.880
Сверхслучайные деревья	False	True	True	SMOTE	0.979655	0.980071	0.979714	*n_estimators = 500 criterion = 'entropy' max_depth = 8 max_features = None*	0.880	0.880

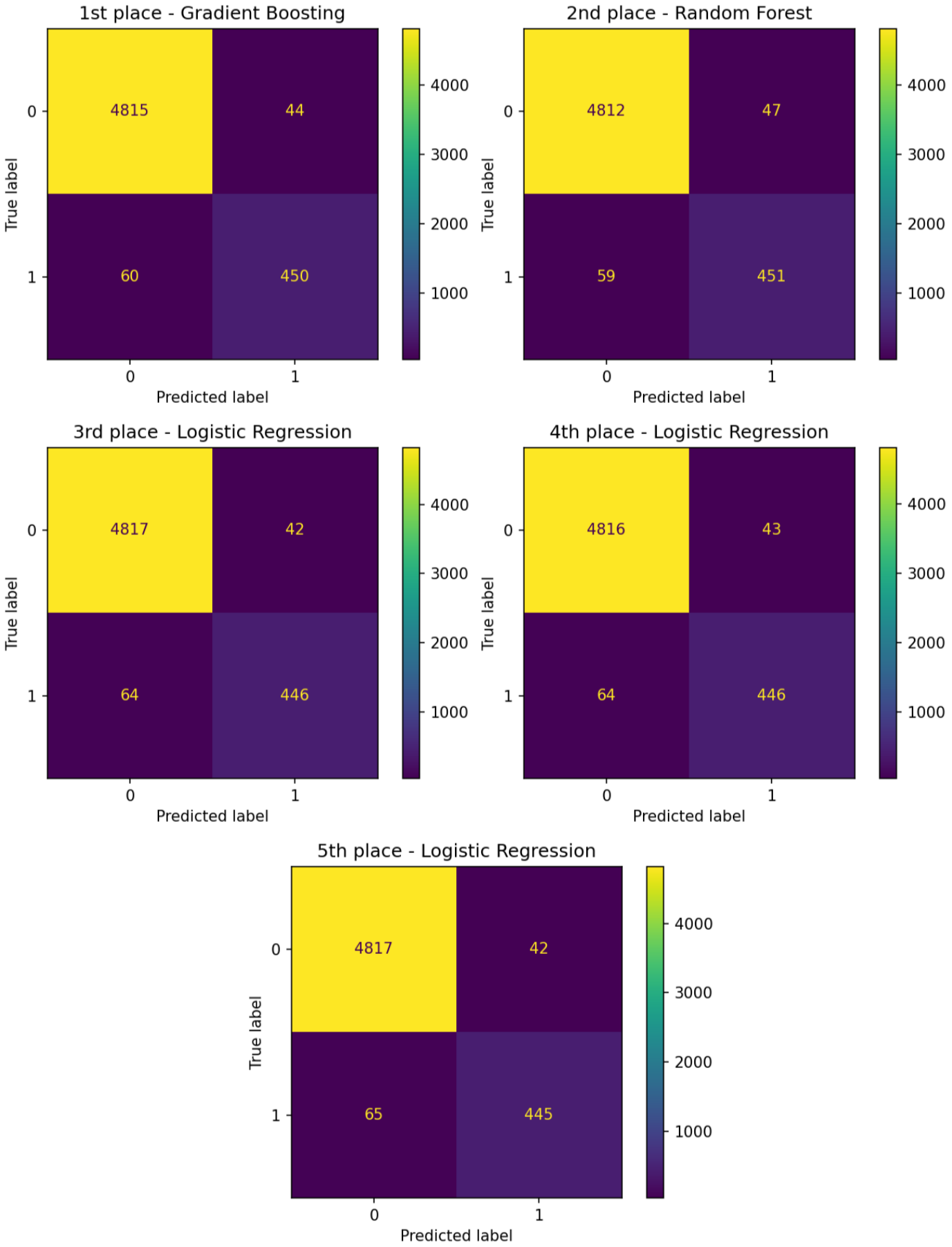

Рисунок 11а. Матрицы ошибок

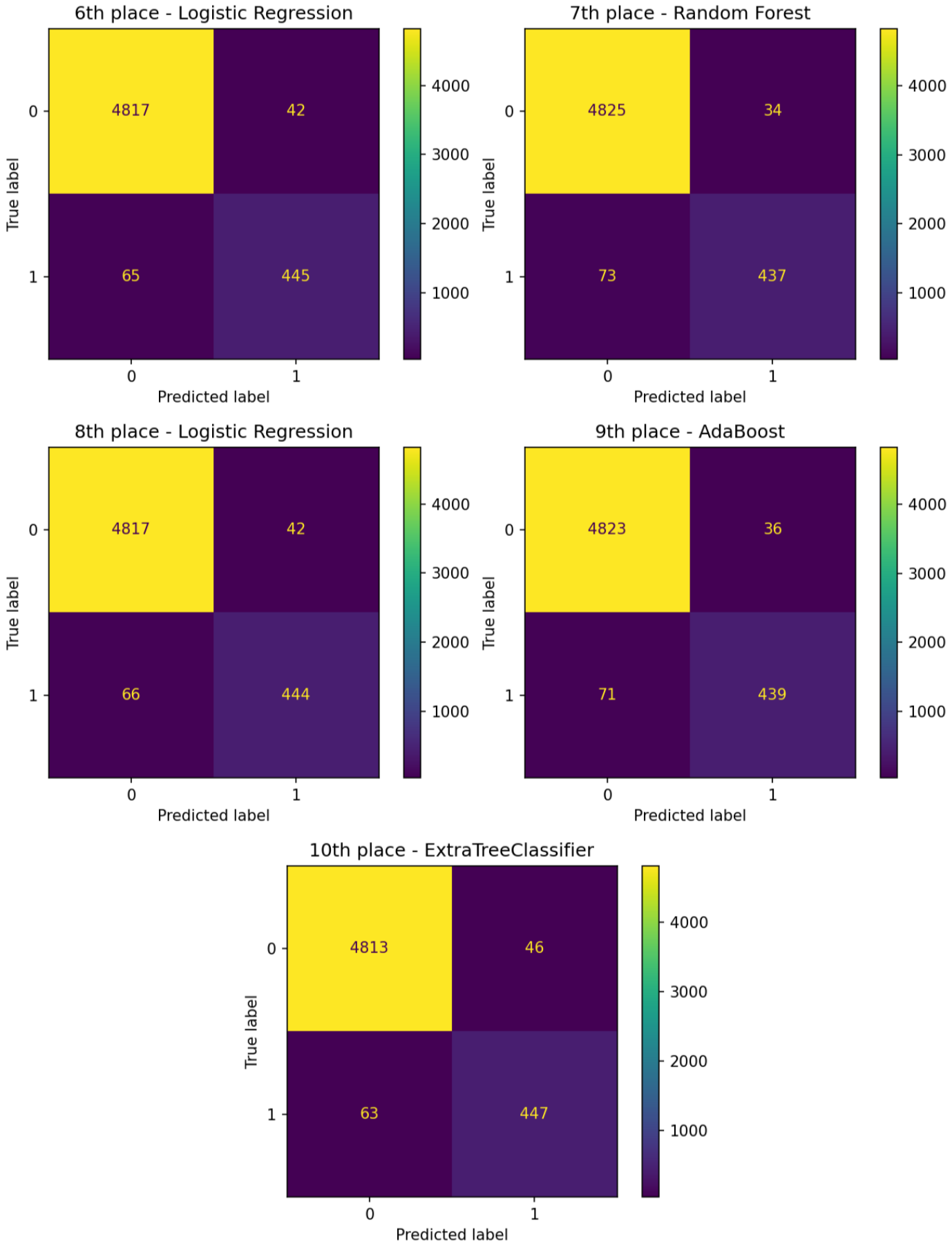

Рисунок 121б. Матрицы ошибок

Как указывалось в самом начале, для нас важно не упустить пульсары, предсказав для них неправильный класс. Те объекты, которые модель классифицировала как пульсар, будут отправлены на проверку человеку, поэтому на матрицах ошибок необходимо обратить внимание именно на два нижних квадрата – количество упущенных и верно предсказанных пульсаров.

Модели, занявшие первые два места, демонстрируют похожие результаты. Модель с использованием градиентного бустинга вырвалась вперёд за счёт большего количества верно классифицированных не-пульсаров. Занявший второе место случайный лес относит к пульсарам три лишних объекта, но при этом верно классифицирует на один пульсар больше, что в данном случае имеет наибольшее значение. Таким образом можно сделать вывод, что использовать рекомендуется именно эту модель.

Заключение

Целью работы было построить наилучшую модель машинного обучения для бинарной классификации звёзд на основе набора данных HTRU2. Рассмотрены метод главных компонент и метод устойчивого скалирования. Дан обзор методам субдискретизации и передискретизации. Рассмотрены восемь алгоритмов машинного обучения, а также популярные метрики оценки качества бинарной классификации. Произведен подбор гиперпараметров методом поиска по сетке. Лучший результат для данной задачи показала модель случайного леса с применением метода устойчивого сэмплирования и передискретизации SMOTE.