Итоговый отчёт

Тема: Проведение исследований с различными алгоритмами машинного обучения (KNN, Linear/Logistic, Decision Tree, Random Forest, Gradient Boosting) на двух задачах: регрессия (Seoul Bike — предсказание Rented Bike Count) и классификация (Details — предсказание DefectStatus)

Автор: Пирязев Михаил **Дата:** 14.12.2025

Оглавление

1. Данные и цель

1.1 Описание задач

Seoul Bike (регрессия)

- Целевая переменная: `Rented Bike Count` (количество арендованных велосипедов)
- Задача: прогноз на основе погодных, временных и иных признаков
- Метрики: MAE, RMSE, R^2

Details (классификация)

- Целевая переменная: `DefectStatus` (0 — норма, 1 — дефект)
- Задача: бинарная классификация дефектного статуса детали по производственным характеристикам
- Метрики: Accuracy, F1-score

1.2 Цели работ

Для каждой лабораторной работы: Построить baseline-модель с минимальным набором признаков, выдвинуть гипотезы о полезных признаках, провести Feature Engineering и предобработку, оптимизировать гиперпараметры моделей, сравнить результаты sklearn-реализаций с собственными алгоритмами.

2. Общая методология

Все исследования следовали единому методологическому подходу:

Baseline – простые модели с минимальным набором признаков

Выдвижение гипотез – предположения о наиболее полезных признаках (временные признаки, лаги, ratio, трансформации)

Feature Engineering – циклическое кодирование времени (sin/cos), лаги и rolling-агрегаты, ratio/KPI, лог-преобразования, взаимодействия между признаками

Предобработка и масштабирование – StandardScaler, QuantileTransformer, OneHotEncoder, где необходимо

Гиперпараметризация – GridSearchCV, RandomizedSearchCV, k-fold CV, временные сплиты для учёта временной структуры данных

Сравнение моделей – sklearn-реализации против собственных, анализ метрик (MAE, MSE, RMSE, R^2 для регрессии; Accuracy, F1 для классификации)

3. Ключевые результаты по лабораторным работам

3.1 Лабораторная 1 — K-Nearest Neighbors (KNN)

Задача	Baseline	Улучшено
Bike (регрессия) — MAE	181.09	173.562
Bike — RMSE	294.81	287.17
Bike — R^2	0.7914	0.8021
Details (классификация) — Accuracy	0.8772	0.8858
Details — F1	0.9325	0.9354

KNN показал стабильные результаты и служит хорошим baseline. Feature Engineering (обработка признаков, масштабирование) положительно повлияли на метрики.

3.2 Лабораторная 2 — Linear / Logistic Regression

Задача	Baseline	Улучшено
Bike (регрессия) — MAE	325.352	317.685
Bike — RMSE	434.902	426.822
Bike — R^2	0.545	0.5628
Details (классификация) — Accuracy	0.875	0.877
Details — F1	0.929	0.929

Линейные модели дают стабильный, но не лучший результат. Модели восприимчивы к масштабированию и выбросам.

3.3 Лабораторная 3 — Decision Tree

Задача	Baseline	Улучшено (sklearn)
Bike (регрессия) — MAE	~176.8	176.77
Bike — MSE	223285	85071.46
Bike — R^2	0.46	0.795
Details (классификация) — Accuracy	0.75	0.961
Details — F1	0.85	0.977

Decision Tree показал значительное улучшение при тюнинге гиперпараметров. Высокая интерпретируемость результатов.

3.4 Лабораторная 4 — Random Forest

Задача	Baseline	Улучшено
Bike (регрессия) — MAE	~284.96	152.92
Bike — RMSE	415.32	272.90
Bike — R^2	0.584	0.821
Details (классификация) — Accuracy	0.816–0.846	0.9606
Details — F1	0.899–0.458	0.9771

Random Forest — ансамблевый метод, показал значительный прирост качества. Параллелизация и усреднение множества деревьев повышают обобщающую способность. MAE снизился почти вдвое.

3.5 Лабораторная 5 — Gradient Boosting

Задача	Baseline	Улучшено
Bike (регрессия) — MAE	348	41.7
Bike — R^2	0.41	0.987
Details (классификация) — Accuracy	0.84	0.951
Details — F1	0.91	0.971

Gradient Boosting показал лучшие результаты среди всех методов. Последовательное исправление ошибок предыдущих моделей даёт наиболее точные прогнозы. R^2 достиг 0.987 — отличный результат.

4. Сравнение алгоритмов машинного обучения

4.1 K-Nearest Neighbors (KNN)

KNN — это алгоритм lazy learning: он не строит явную модель на этапе обучения, а формирует прогноз только в момент запроса, находя K ближайших объектов в пространстве признаков (обычно по евклидову расстоянию). Его плюсы в простоте и понятности, отсутствии отдельного этапа обучения, универсальности для задач регрессии и классификации, а также в достойной эффективности на небольших наборах данных. Минусы — медленное предсказание на больших выборках из-за необходимости считать расстояния до множества точек, чувствительность к масштабу признаков (поэтому требуется нормализация), проблемы в высоких размерностях из-за «проклятия размерности», необходимость хранить всю обучающую выборку в памяти и уязвимость к шуму и выбросам при неудачном выборе K и метрики. Использовать KNN разумно как быстрый baseline для проверки гипотез, при умеренных объёмах данных (условно до ~100k наблюдений с

оговорками по числу признаков) и когда нужно поймать локальные нелинейные зависимости, которые не видят линейные модели.

4.2 Линейная и логистическая регрессия

Линейная регрессия (Linear Regression) применяется в задачах регрессии, когда нужно предсказывать непрерывные значения, и при этом предполагается, что зависимость целевой переменной от признаков близка к линейной: $y = w_0 + w_1 x_1 + \cdots + w_n x_n$. Параметры такой модели можно находить либо аналитически (через нормальное уравнение), либо итерационно — например, градиентным спуском. Логистическая регрессия (Logistic Regression), несмотря на название, используется уже для классификации: она берёт линейную комбинацию признаков и превращает её в вероятность принадлежности к классу с помощью сигмоиды $p = \frac{1}{1+e^{-z}}$, поэтому модель остаётся интерпретируемой и даёт удобный вероятностный выход.

Плюсы линейной и логистической регрессии в том, что они обычно быстро обучаются и быстро делают предсказания, хорошо масштабируются на больших наборах данных, требуют мало памяти и часто выступают сильным baseline; кроме того, коэффициенты модели позволяют довольно прозрачно понимать вклад каждого признака. Основные минусы связаны с их «линейной природой»: они предполагают линейную связь (или линейную разделимость в случае классификации), поэтому плохо ловят сложные нелинейные закономерности без дополнительной инженерии признаков; также на практике они нередко чувствительны к масштабу признаков, из-за чего полезно применять нормализацию/стандартизацию. Использовать такие модели стоит, когда нужно быстро собрать baseline, когда важна интерпретируемость, когда данные примерно линейно устроены (или близки к линейно разделимым), а также когда требуется именно вероятностный выход для принятия решений.

4.3 Decision Tree (дерево решений)

Дерево решений строит иерархию правил вида «если признак A больше (или меньше) некоторого значения, то идём по одной ветке, иначе — по другой». На каждом шаге выбирается такое разбиение, которое лучше всего «очищает» узлы: обычно за счёт максимизации информационного выигрыша или минимизации примеси (например, по критерию Gini), пока не будет достигнуто условие остановки. Главные плюсы дерева — высокая интерпретируемость (логику решения легко объяснить), отсутствие необходимости в масштабировании признаков, способность улавливать нелинейные зависимости и взаимодействия, а также достаточно быстрое предсказание. Во многих реализациях дерево может работать с категориальными признаками напрямую или после простого кодирования, что тоже удобно в прикладных задачах.

Основные минусы связаны с тем, что одиночное дерево легко переобучается и может «запоминать» обучающую выборку, поэтому без ограничений глубины и размеров листьев качество на новых данных часто падает. Кроме того, деревья неустойчивы: небольшие изменения в данных могут привести к совсем другой структуре, и при добавлении новых данных модель нередко приходится перестраивать заново, поэтому важен аккуратный тюнинг параметров вроде max_depth и min_samples_split. Использовать дерево решений стоит, когда важна объяснимость, в данных есть смесь числовых и категориальных признаков, объём данных не слишком большой и требуется понятная логика принятия решений (например, для отчётности или согласования с бизнесом).

4.4 Random Forest (случайный лес)

Random Forest — это ансамблевый метод, который строит N деревьев решений на случайных подвыборках объектов и случайных подмножествах признаков, а итоговый ответ получает объединением предсказаний отдельных деревьев: усреднением в регрессии и голосованием в классификации. За счёт такого

«усреднения» ошибок модель обычно заметно меньше переобучается по сравнению с одиночным деревом, при этом хорошо держит качество на средних и больших датасетах, умеет ловить нелинейные зависимости и в целом довольно устойчива к шуму и выбросам. Дополнительно Random Forest часто используют из-за встроенной оценки важности признаков и потому, что обучение деревьев можно распараллеливать.

При этом Random Forest менее интерпретируем: уже нельзя так же прозрачно объяснить решение, как в одном дереве, потому что финальный ответ — это агрегат по множеству моделей. Также он обычно обучается и предсказывает медленнее, чем линейные модели, потребляет больше памяти, а при дисбалансе классов может сильнее «тянуться» к большинству, если не делать балансировку или настройку весов. Использовать Random Forest стоит, когда данных уже достаточно много, нужен хороший баланс между точностью и объяснимостью (пусть и не идеальной), в задаче много нелинейностей и смешанные типы признаков, а ещё когда полезно получить понятные оценки важности признаков для дальнейшего feature engineering или отбора фич.

4.5 Gradient Boosting (градиентный бустинг)

Градиентный бустинг — это последовательный ансамбль, где деревья строятся не независимо, а одно за другим: каждое новое дерево обучается так, чтобы исправлять ошибки (то есть уменьшать функцию потерь) предыдущей композиции моделей. Модель добавляет деревья с небольшим вкладом, который контролируется скоростью обучения (learning rate), поэтому качество улучшается итеративно и обычно довольно стабильно, если правильно настроить регуляризацию. Популярные реализации этого подхода — XGBoost, LightGBM, CatBoost, а в sklearn часто используют HistGradientBoosting, который оптимизирован под табличные данные и большие выборки. Главное преимущество бустинга — то, что на табличных данных он очень часто даёт лучшую итоговую точность среди «классических» методов, а

также позволяет гибко подстраиваться под разные функции потерь и задачи (регрессия/классификация). При этом у него есть встроенные механизмы регуляризации: можно ограничивать глубину деревьев, вводить shrinkage через learning rate, регулировать количество деревьев и другие параметры, что помогает контролировать переобучение и улучшать обобщающую способность.

Минусы в том, что обучение обычно более тяжёлое по вычислениям и времени, потому что деревья строятся последовательно и полностью распараллелить процесс нельзя, а качество сильно зависит от гиперпараметров — без тюнинга легко получить либо недообучение, либо переобучение. Интерпретируемость тоже ниже, чем у одиночного дерева или линейных моделей, и часто требуется больше памяти и ресурсов, особенно при больших датасетах и активном подборе параметров. Использовать градиентный бустинг стоит, когда важна максимальная точность (например, в соревнованиях и продакшн-задачах на табличных данных), есть ресурсы на обучение и настройку, и уже проделана базовая работа по feature engineering и проверке гипотез более простыми моделями.

5. Что именно давало наибольший прирост качества

5.1 Feature Engineering

В задаче Seoul Bike сильнее всего помогли временные признаки: часы были закодированы циклически через sin/cos, чтобы модель «понимала» периодичность суток, дополнительно были добавлены месяц, день недели, признаки праздников и индикатор пиковых/непиковых часов — в итоге заметно снизился RMSE, особенно на часах пиковой нагрузки. Дальше качество усилили лаги и скользящие агрегаты: в признаки добавлялись значения проката за предыдущие часы (например, лаг 1–2), а также rolling-метрики вроде среднего за 6 и 24 часа и стандартного отклонения в окне; это дало модели контекст прошлых состояний и привело к улучшению MAE и RMSE примерно на 15–25% за счёт лучшего учёта краткосрочных трендов и сезонности. Для классификации Details ощутимый вклад дали ratio и KPI-признаки: разные отношения параметров (например, Cost_per_Unit и Defect_per_Unit) и индексы качества помогали лучше отделять аномальные случаи от нормальных,

потому что такие нормированные показатели часто устойчивее «сырых» величин. Дополнительно применялось лог-преобразование вида log(признак + 1) для скошенных распределений — это снижало влияние выбросов и делало обучение как линейных моделей, так и деревьев более стабильным. Наконец, добавление взаимодействий признаков (например, temp × humidity, wind × precipitation) и при необходимости полиномиальных комбинаций помогало захватывать нелинейные зависимости, которые простые модели без таких преобразований обычно не видят.

5.2 Подбор гиперпараметров

GridSearchCV и RandomizedSearchCV — это подходы к подбору гиперпараметров, где модель обучается и оценивается на разных комбинациях настроек: в GridSearchCV перебор идёт систематически по заданной сетке значений, а в RandomizedSearchCV случайно выбираются варианты из распределений или списка. Чтобы оценка качества была честной и устойчивой, обычно используют k-fold кросс-валидацию, поэтому итоговые параметры выбираются не по одному разбиению, а по среднему качеству на нескольких фолдах. На практике такой тюнинг заметно улучшает обобщающую способность модели и часто даёт прирост метрик примерно на 1–10% за счёт более удачного баланса между недообучением и переобучением.

6. Выводы

Feature engineering оказался главным драйвером качества: тщательно продуманные признаки почти всегда дают больший прирост метрик, чем просто замена модели на более сложную. На фоне базовых подходов вроде KNN и линейных моделей ансамблевые методы в целом показали себя сильнее: и Random Forest, и Gradient Boosting стабильно давали более высокое качество в обеих задачах. При этом градиентный бустинг стал лидером по точности — при грамотном наборе признаков и тюнинге он вышел на лучшие результаты (например, на Seoul Bike удалось получить $R^2 = 0.987$). Самописные реализации оказались полезны именно как учебная часть: они помогают реально понять, как работают алгоритмы, но в реальных проектах разумнее опираться на оптимизированные библиотечные решения. Наконец, для временных рядов критически важна time-aware валидация: случайный train/test split легко даёт утечку «будущей» информации и из-за этого приводит к завышенным, нечестным оценкам качества.