Теоретический минимум второй лабораторной по машинному обучению

---

1. Обучение с учителем vs обучение без учителя

- **С учителем(*Supervised)***: У нас есть данные с метками (ответами). Модель учится предсказывать эти метки по входным данным.
  Примеры:
  - **Классификация** (предсказать категорию, например, "кошка" или "собака" на фото).
  - **Регрессия** (предсказать число, например, цену дома).
  - **Ранжирование** (упорядочить объекты, например, выдать список релевантных товаров для поиска).
  - **Генерация** (создать новый контент на основе примеров, например, генерация текста с указанием темы).
- **Без учителя(*Unsupervised)***: Данные без меток. Модель сама ищет структуры или закономерности.
  Примеры:
  - **Кластеризация** (группировать данные по сходству, например, разделить клиентов на сегменты без предварительных меток).
  - **Генерация** (создать новые данные, например, генерация изображений без указания конкретных тем — как в GAN).

Чем отличаются?

- С учителем: "Учитель" даёт правильные ответы, модель учится их повторять.
- Без учителя: "Учителя нет", модель сама находит скрытые паттерны и закономерности (например, группы похожих объектов).

---

2. Параметры модели vs гиперпараметры

- **Параметры**: внутренние величины, которые обучаются моделью. *Пример*: В линейной регрессии `y = a*x + b` — это a и b (веса).
- **Гиперпараметры**: Настройки, которые **задаёт человек** до обучения. Они влияют на то *как* модель учится, но не участвуют в предсказании.
  Пример: Глубина дерева решений, скорость обучения, количество соседей в KNN.

---

3. Зачем перебирать гиперпараметры?

Чтобы найти лучшие оптимальные настройки для модели, которые дадут наилучшие результаты на тестовых данных. Например, если глубина дерева слишком большая — модель переобучится, слишком маленькая — модель не выучит закономерности.

Виды переборов: Полный перебор, Случайный перебор, Байесовская оптимизация (более сложный метод, который не предполагает, что функция монотонна. Строит вероятностную модель целевой функции, что позволяет быстрее находить хорошие решения в сложных пространствах параметров.)

Перебор (например, через GridSearchCV) позволяет подобрать оптимальные значения. Также есть RandomizedSearchCV и ручной перебор.

---

4. Мягкая классификация

Когда модель предсказывает **вероятности принадлежности к классам**, а не жёсткую метку.
Пример: Для изображения кошки модель говорит "90% кошка, 10% собака". Это полезно, когда нужно оценить уверенность предсказания (например, в медицине: "70% вероятность болезни" — лучше, чем "болезнь есть/нет").

---

5. Переобучение (overfitting)

Когда модель **запоминает шум и детали обучающих данных**, а не учится общим закономерностям. Результат: отлично работает на тренировочных данных, но плохо на новых.
Пример: Модель, которая точно предсказывает цену дома по номеру квартиры в тренировочном наборе, но не справляется с новыми домами.
Причины: Слишком сложная модель, мало данных или слишком долгое обучение.

Как бороться? Регуляризация, упрощение модели, увеличение данных, кросс-валидация.

---

6. Бейзлайн и наивные алгоритмы

- **Бейзлайн**: Простой алгоритм, который служит отправной точкой для сравнения. Задает минимальный уровень точности или качества, который нужно превзойти. Если ваша модель хуже бейзлайна — она бесполезна.
- **Наивный алгоритм**: Очень простая модель, которая часто работает "наивно" (не учитывает сложных закономерностей и использует самые простые правила для предсказания), но даёт базовое качество.

Примеры:

- Для классификации – всегда предсказывать самый популярный класс.
- Для регрессии – всегда предсказывать среднее/медиану выборки.

---

7. Дискретизация, бинаризация, нормализация, взвешивание признаков

- **Дискретизация**: Преобразование непрерывных признаков в категории.
  Зачем? Для моделей, которые плохо работают с непрерывными данными (например, деревья решений). Упрощает анализ, уменьшает шумы, позволяет захватить нелинейные зависимости.

  Пример: Возраст → группы "0-18", "19-35", "36-60".
- **Бинаризация**: Превращение признака в 0/1.
  Зачем? Для упрощения или если признак имеет только два состояния (например, "есть ли кредитная карта: да/нет").
- **Нормализация**: Приведение признаков к одному масштабу (например, [0, 1] или среднее=0, дисперсия=1).
  Зачем? Модели вроде KNN или нейросети чувствительны к масштабу. Если один признак в тысячу раз больше другого — он будет доминировать. Устраняет влияние выбросов с большими значениями. ускоряет обучение.
- **Взвешивание признаков**: Добавление весов для признаков (например, важность "возраст" выше, чем "цвет волос").
  Зачем? Чтобы модель больше внимания уделяла важным признакам. Упрощает работу модели, улучшает точность, уменьшает влияние несущественных признаков.

---

8. One-hot encoding и binary encoding – варианты бинаризации

- **One-hot encoding**: Превращение категориального признака в набор бинарных столбцов.
  Пример: Цвет → ["красный": 1, 0, 0], ["зелёный": 0, 1, 0], ["синий": 0, 0, 1].
  В pandas: Делается через `pd.get_dummies()`
- **Binary encoding**: Категории кодируются битами (например, 3 категории → 2 бита: 00, 01, 10).
  Чем отличается: One-hot создаёт много новых столбцов (количество категорий), binary — компактнее ($\log_2$(количество категорий)), но сложнее интерпретировать.

---

9. Cross-entropy и почему она "cross"

- **Что это**: Мера расхождения между **предсказанными вероятностями** и **истинными метками**. Так называемая функция потерь.
  Формула: `-sum(истинная_метка * log(предсказанная_вероятность))`.
- **Почему "cross"?** Потому что сравнивает два распределения:
  - Истинное распределение (метки),
  - Предсказанное распределение (вероятности модели).

Слово "cross" отражает, что мы "пересекаем" эти два распределения для измерения ошибки.

---

10. Деление данных: train, test, val

- **Train**: Для обучения модели (большая часть данных).
- **Val (валидация)**: Для настройки гиперпараметров и оценки качества во время обучения. Используется для выбора лучшей модели. После каждой эпохи обучения модель тестируется на Val.
- **Test**: Для финальной оценки модели *после* всех настроек. Данные не должны использоваться ни при обучении, ни при валидации.
- **Кросс-валидация (k-fold)**: Данные делятся на k частей. По очереди одна часть — валидация, остальные — обучение. Повторяется k раз. Уменьшает случайность деления данных.
- **Data leak**: Когда информация из теста/валидации "просачивается" в обучение.
  Пример: Нормализация всего датасета перед разделением на train/test — тогда статистики теста влияют на обучение.

---

11. Регуляризация: L1 и L2

- **Зачем?** Чтобы избежать переобучения, "штрафуя" сложные модели. Этот штраф препятствует слишком большим значениям параметров (весов), заставляя модель быть более гладкой и менее чувствительной к шуму в данных.
- **L1 (Lasso)**: Добавляет штраф за сумму абсолютных значений весов.
  Эффект: Обнуляет незначимые веса (отбор признаков). Модель может автоматически выбрать наиболее важные признаки, а остальные зануляются.
- **L2 (Ridge)**: Добавляет штраф за сумму квадратов весов.
  Эффект: Смягчает веса, но не обнуляет их. Делает модель более устойчивой к шуму.

---

12. Метрики классификации

- **Confusion matrix**: Таблица с 4 числами:
  - True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN).
- **Accuracy**: `(TP + TN) / всего`. Доля правильных ответов/предсказаний среди всех предсказаний. *Минус*: не работает при дисбалансе классов.
- **Precision**: `TP / (TP + FP)`. Доля правильных положительных предсказаний среди всех предсказанных.
  Пример: "Из всех пациентов, которых модель назвала больными, сколько действительно больны?"
- **Recall (Sensitivity)**: `TP / (TP + FN)`. Доля реальных положительных, которые модель нашла.
  Пример: "Из всех больных, сколько она нашла?"
- **F1**: Среднее гармоническое precision и recall (`2 * (precision * recall) / (precision + recall)`). Баланс между ними.
- **F-beta**: Вариант F1 с весом recall (если beta > 1, важнее recall).
- **ROC AUC**: Площадь под кривой, показывающая способность модели различать классы. По осям ROC кривой откладывается:
  - False Positive Rate (FPR) = FP / (FP + TN) (по оси X)
  - True Positive Rate (TPR) = Recall = TP / (TP + FN) (по оси Y)

  Как строится: Меняем порог вероятности для классификации, считаем True Positive Rate (TPR) и False Positive Rate (FPR) для каждого порога.
- **Precision-Recall кривая**: Зависимость precision от recall при изменении порога. Полезна при дисбалансе классов.

---

13. Метрики регрессии

- **MAE (Mean Absolute Error)**: Среднее абсолютное отклонение предсказаний от реальных.
  Формула: `sum(|y_true - y_pred|) / n`.
- **MSE (Mean Squared Error)**: Среднее квадратичное отклонение. Больше штрафует большие ошибки.
  Формула: `sum((y_true - y_pred)²) / n`.
- **MAPE (Mean Absolute Percentage Error)**: Средняя абсолютная процентная ошибка.
  Формула: `sum(|(y_true - y_pred)/y_true|) / n * 100%`.
- **SMAPE (Symmetric MAPE)**: Симметричная версия MAPE, чтобы избежать проблем с нулевыми значениями. Полностью нормированна.

---

14. Можно ли использовать ML для заполнения пропусков в датасете?

Да! Например:

- Для числовых данных: регрессия на другие признаки, KNN (находим ближайших соседей и заполняем средним).
- Для категориальных: мода или классификатор (например, случайный лес).
- Специальные методы: **MICE** (Multiple Imputation by Chained Equations) — строит модели для каждого признака с пропусками, используя другие признаки.

---

15. Дисбаланс классов

- **Как влияет?** Если много экземпляров одного класса и мало другого - искажаются метрики. Метрики вроде accuracy становятся обманчивыми (например, 95% accuracy, но модель всегда предсказывает главный класс). Recall для редкого класса может быть близок к нулю.
- **Как справиться?**
  - Взвешивание классов в модели (например, в логистической регрессии class_weight="balanced").
  - Oversampling**:** Увеличить количество примеров редкого класса (например, генерация синтетических данных через SMOTE).
  - Undersampling**:** Уменьшить количество примеров доминирующего класса.
  - Использовать метрики, которые учитывают дисбаланс (F1, ROC AUC, Precision-Recall).

---

16. Градиентный спуск

- **Зачем?** Минимизировать функцию потерь (ошибку модели).
- **Как работает?** Итеративное обновление параметров модели с целью минимизации функции потерь. Двигаемся в направлении, где ошибка уменьшается быстрее (по градиенту — производной функции потерь).
- **Виды**:
  - **Пакетный (batch)**: Обновляем веса на всей выборке. Медленно, но стабильно.
  - **Мини-батчевый**: Обновляем веса на подмножестве данных (например, 32 примера). Быстрее и стабильнее пакетного.
  - **Стохастический (SGD)**: Обновляем веса на каждом случайном примере. Быстро, но шумно (много колебаний).

---

17. Линейная регрессия

- **Что это**: Строит зависимость между одной целевой переменной (зависимой) и одной или несколькими независимыми переменными (факторами). Предсказывает числовое значение как линейную комбинацию признаков: `y = w1*x1 + w2*x2 + ... + b`
- **Как обучается**: Подбираем коэффициенты, чтобы было похоже на исходную выборку. Минимизирует MSE между предсказаниями и реальными значениями. Получаем прямую с каким то наклоном по которой предсказываем значения.
- **Ограничение**: Не работает, если зависимость нелинейная.

---

18. Логистическая регрессия

- **Что это**: Классификация с помощью линейной модели, но с **логистической функцией** (sigmoid) для превращения результата в вероятность: `p = 1 / (1 + e^(- (w1*x1 + ... + b)))`
- **Зачем sigmoid?** Чтобы выдать вероятность в диапазоне [0, 1].
- **Обучение**: Минимизирует cross-entropy между предсказанными вероятностями и метками.

---

19. Дерево решений и случайный лес

- **Дерево решений**:
  - Графическая модель, состоящая из узлов и ветвей, где каждый узел представляет собой проверку (условие) по одному из признаков, а листья - предсказания (результаты).
  - Решает задачи, разбивая данные на основе признаков (например, "возраст > 30? Да/Нет").
  - **Важность признаков**: Считается по тому, насколько уменьшается неоднородность (например, энтропия) при разделении по признаку. Чем сильнее уменьшение — тем важнее признак.
- **Случайный лес**:
  - Ансамбль деревьев решений. Каждое дерево обучается на случайном подмножестве данных и признаков.
  - **Почему лучше?** Уменьшает переобучение, устойчив к шуму.
  - **Важность признаков**: Среднее значение важности по всем деревьям.

---

20. Ансамблирование, бустинг и другие алгоритмы

- **Ансамблирование**: Объединение нескольких моделей для улучшения качества.
  - **Бэггинг (Bagging)**: Параллельное обучение моделей на случайных подвыборках (например, Random Forest).
  - **Бустинг (Boosting)**: Последовательное обучение моделей, каждая исправляет ошибки предыдущей.
    Примеры: XGBoost, LightGBM, CatBoost.
  - **Стекинг (Stacking)**: Объединение моделей через "мета-модель", которая учится на их предсказаниях.
- **Почему это работает?** Разные модели ошибаются по-разному — объединяя их, компенсируем слабые места.

Оглавление