Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

«Анализ и поиск в больших базах данных»

Направление подготовки 09.04.02 «Информационные системы и технологии»
Направленность (профиль) «Искусственный интеллект, математическое моделирование и суперкомпьютерные технологии в разработке информационных систем»

Ставрополь
2026

Содержание

ВВЕДЕНИЕ

Цель преподавания дисциплины «Анализ и поиск в больших базах данных» состоит в изучение теоретических основ анализа больших данных, включая базовые элементы статистического программирования и интеллектуального анализа больших наборов данных.

Задачи дисциплины – научить производить расчеты с применением технологий анализа больших данных и решать широкий спектр прикладных задач обработки больших наборов данных.

Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК- 2 - способен выбирать, разрабатывать и проводить экспериментальную проверку работоспособности программных компонентов систем, основанных на знаниях, по обеспечению требуемых критериев эффективности и качества функционирования	ПК-$2_{ИД-2.1}$ - выбирает и разрабатывает программные компоненты систем, основанных на знаниях	Осуществляет выбор способов экспериментальной проверки работоспособности программных компонентов систем, основанных на знаниях, по обеспечению требуемых критериев эффективности и качества функционирования Разрабатывает методы экспериментальной проверки работоспособности программных компонентов систем, основанных на знаниях, по обеспечению требуемых критериев эффективности и качества функционирования Применяет навыки проведения экспериментальной проверки работоспособности программных компонентов систем, основанных на знаниях, по обеспечению требуемых критериев эффективности и качества функционирования
	ПК-$2_{ИД-2.2}$ - проводит экспериментальную проверку работоспособности систем, по обеспечению требуемых критериев эффективности и качества функционирования,.	Осуществляет экспериментальную проверку работоспособности систем по обеспечению по обеспечению требуемых критериев эффективности Имеются знания об основных методах экспериментальной проверки работоспособности систем Осуществляет качества функционирования экспериментальную проверку работоспособности систем
ПК-3 Способен выбирать и применять методы инженерии знаний для созда-	ПК-$3_{ИД-3.1}$ - выбирает и применяет методы сбора и извлечения знаний	Демонстрирует знания методов инженерии знаний для создания систем, основанных на знаниях Выбирает методы инженерии знаний для создания систем, основанных на знаниях

ния систем, основанных на знаниях		Применяет методы инженерии знаний для создания систем, основанных на знаниях
	ПК-3$_{ИД-3.2}$ - использует методы инженерии знаний для создания систем, основанных на знаниях	Демонстрирует знания о методах инженерии знаний для создания систем Применяет методы инженерии знаний для создания систем, основанных на знаниях
ПК-8 Способен осуществлять руководство по созданию и развитию систем и комплексов обработки данных, в том числе больших данных, для корпоративных и государственных заказчиков	ПК-8$_{ИД-8.1}$ - участвует в создании (модернизации) общедоступных платформ для хранения наборов данных, соответствующих методологиям описания, сбора и разметки данных; хранения наборов данных (в том числе звуковых, речевых, медицинских, метеорологических, промышленных данных и данных систем видеонаблюдения) на общедоступных платформах для обеспечения потребностей организаций разработчиков в области искусственного интеллекта	Имеются знания о методологиях описания, сбора и разметки данных Участвует в создании общедоступных платформ для хранения наборов данных, соответствующих методологиям описания, сбора и разметки данных Модернизирует общедоступные платформы для хранения наборов данных, соответствующих методологиям описания, сбора и разметки данных Разрабатывает информационные системы хранения наборов данных (в том числе звуковых, речевых, медицинских, метеорологических, промышленных данных и данных систем видеонаблюдения) на общедоступных платформах для обеспечения потребностей организаций разработчиков в области искусственного интеллекта
	ПК-8$_{ИД-8.2}$ - осуществляет руководство по созданию и развитию систем и комплексов обработки данных, в том числе больших данных для корпоративных и государственных заказчиков	Осуществляет руководство по созданию систем обработки данных, в том числе больших данных для корпоративных заказчиков Владеет навыками создания систем обработки данных, в том числе больших данных для государственных заказчиков Руководит работами по созданию комплексов обработки данных, в том числе больших данных для корпоративных заказчиков Применяет навыки развития комплексов обработки данных, в том числе больших данных для государственных заказчиков
ПК-9 Способен руководить проектами по созданию комплексных систем на основе аналитики больших данных в различных отраслях	ПК-9$_{ИД-9.1}$ - осуществляет руководство проектом по построению комплексных систем на основе аналитики больших данных в различных отраслях	Осуществляет подбор команды для решения задач построения комплексных систем на основе аналитики больших данных в различных отраслях Имеется знания о методах технологиях и средствах разработки комплексных систем на основе аналитики больших данных в различных отраслях Руководит проектом по построению комплексных систем на основе аналитики больших данных в различных отраслях

	ПК-9$_{ИД-9.2}$ - руководит проектами по созданию комплексных систем в различных отраслях	Осуществляет выбор членов команды для разработки систем Разрабатывает техническое задание проекта Руководит проектами по созданию комплексных систем в различных отраслях
ПК-12 Способен разрабатывать и исследовать теоретические и экспериментальные модели объектов профессиональной деятельности на основе искусственного интеллекта, математического моделирования и суперкомпьютерных технологи	ПК-12$_{ИД-12.1}$ - разрабатывает методику выполнения аналитических работ в контексте исследования модели объектов профессиональной деятельности на основе методов математического моделирования и искусственного интеллекта	Имеются знания методов математического моделирования и искусственного интеллекта Разрабатывает методику выполнения аналитических работ в контексте исследования модели объектов профессиональной деятельности Применяет методы математического моделирования и искусственного интеллекта для разработки модели объектов профессиональной деятельности
	ПК-12$_{ИД-12.2}$ - планирует и организует аналитические работы в информационно-технологическом проекте на основе суперкомпьютерных технологий и методов искусственного интеллект	Демонстрирует знания суперкомпьютерных технологий для планирования аналитические работы в информационно-технологическом проекте Имеются знания о методах интеллекта, использующихся при выполнении аналитических работ Планирует аналитические работы в информационно-технологическом проекте Организует аналитические работы в информационно-технологическом проекте

Лабораторная работа 1

Метрики качества задач классификации

Цель: получение знаний основных метрик качества бинарной классификации и вариантов тонкой настройки алгоритмов классификации.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.

Вопросы для обсуждения на лабораторном занятии: алгоритмы классификации и их сравнительный анализ.

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

Примеры алгоритмов классификации

Для расчета метрик качества в задаче машинного обучения с учителем необходимы только две величины: векторы действительных и предсказанных значений. Действительные значения – это метки классов в тренировочной и тестовой выборке; алгоритм классификации возвращает предсказанные.

Рассмотрим несколько получаемых на выходе модели примеров векторов.

Пусть в нашей задаче действительные значения составляют вектор из нулей и единиц, а предсказанные лежат в интервале (где число означает вероятность отнести пример к классу “1”). Такие пары векторов будем визуализировать так, как представлено на рисунке 1.1.

Чтобы сделать финальное предсказание (отнести пример к классу “0” или“1”), нужно установить порог (горизонтальная линия на рисунке 1.1): всем объектам с предсказанным значением выше будет присвоен класс “1”, остальным – “0”.

Наилучшая ситуация: порог абсолютно верно разделяет предсказанные вероятности по двум классам. Для примера с рис. 1.2а идеальные интервалы вероятностей можно получить путем выбора порога .

Чаще всего вероятностные интервалы накладываются друг на друга (рисунок 1.2б) – тогда приходится подбирать порог внимательно.

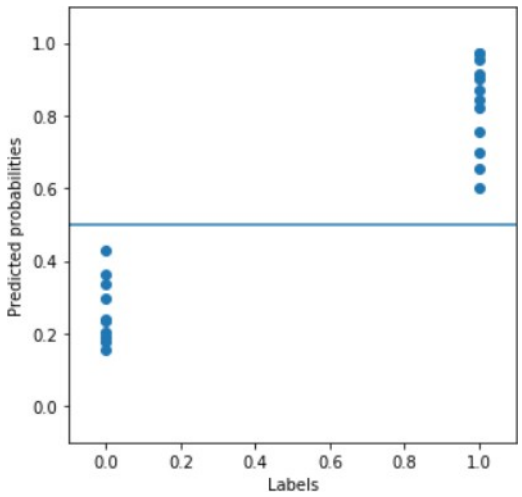
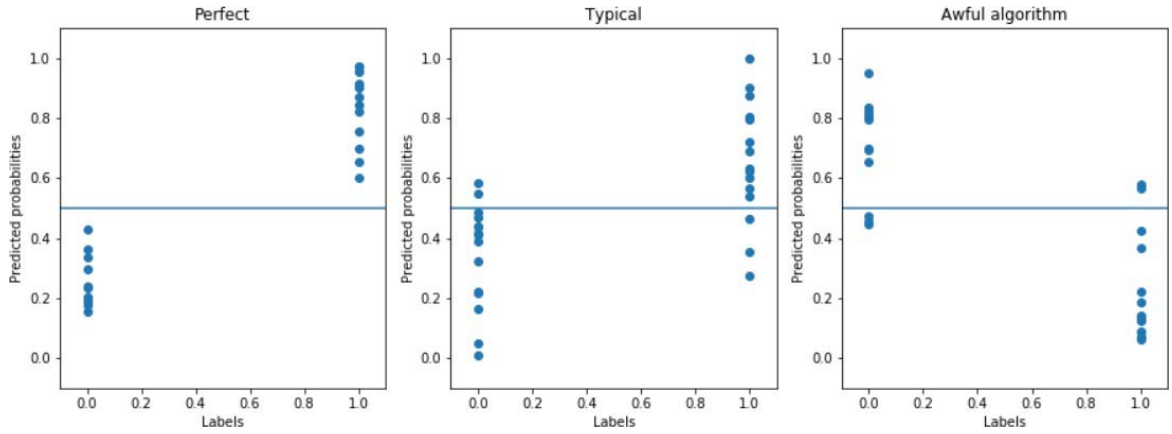


Рисунок 1.1 – Пример действительных (по оси абсцисс) и предсказанных (по оси ординат) значений

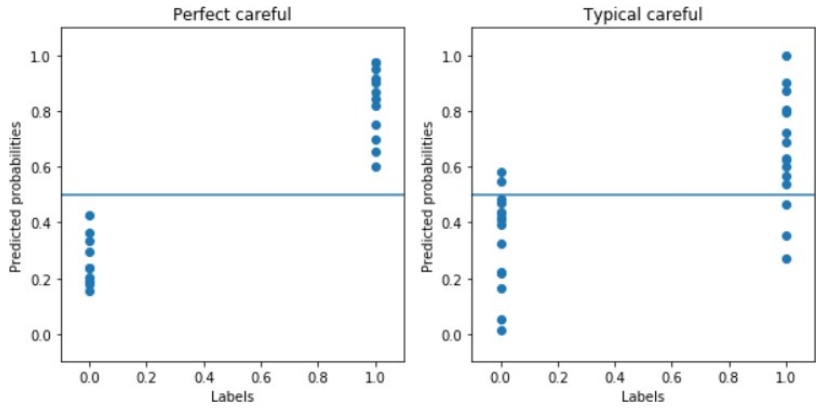
Неверно обученный алгоритм совершает обратное: он ставит вероятности объектов класса “0” выше вероятностей примеров класса “1” (рисунок 1.2в). В такой ситуации следует проверить, не были ли перепутаны метки “0” и “1” при получении тренировочной выборки.



а б в

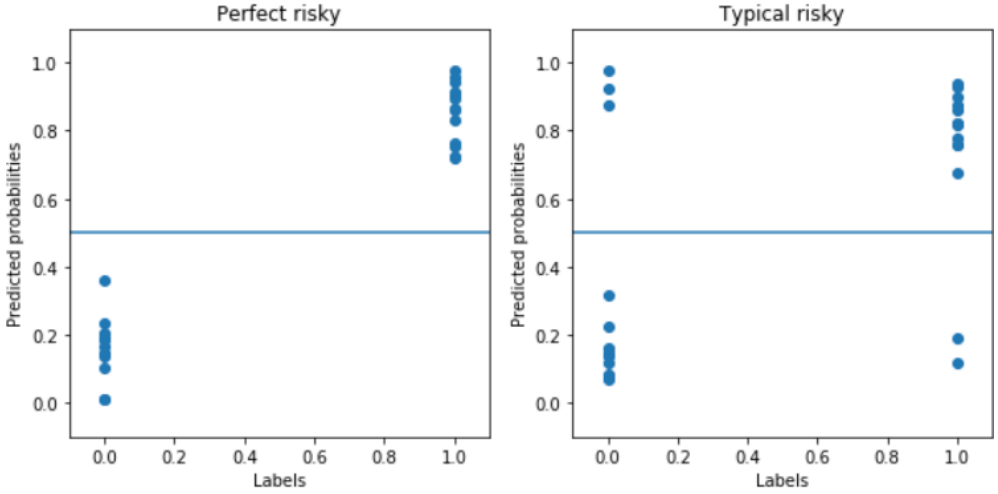
Рисунок 1.2 – Примеры действительного и предсказанного векторов для идеального (а), типичного (б) и неверно обученного (в) алгоритмов

Алгоритмы могут быть осторожными и выдавать значения, не слишком отдаленные от 0,5 (рис. 1.3), или могут брать на себя риск по получению значений, близких к нулю и единице (рис. 1.4).



а б

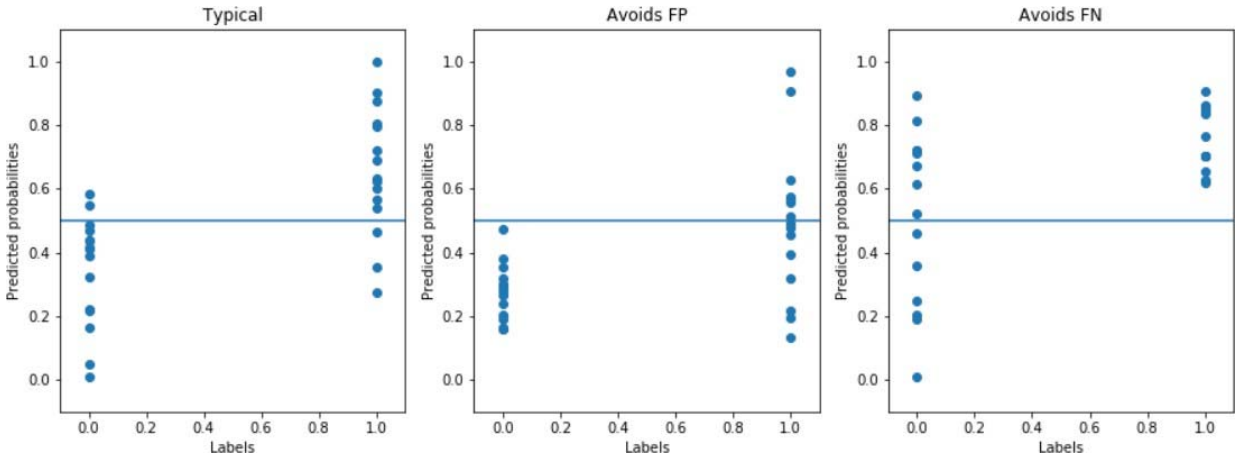
Рисунок 1.3 – Примеры действительного и предсказанного векторов для идеального (а) и типичного (б) осторожного алгоритма



а б

Рисунок 1.4 – Примеры действительного и предсказанного векторов для идеального (а) и типичного (б) рискующего алгоритма

Интервалы вероятностей могут быть смещены. Например, еслинежелательны ошибки I рода (false-positive), то алгоритм будет выдавать значения в среднем ближе к нулю. Аналогично, для избегания ошибок II рода (false-negative) чаще необходимо получать на выходе модели вероятности выше0,5. На рисунке 1.5 представлены примеры векторов в данных ситуациях.



а б в

Рисунок 1.5 – Примеры действительного и предсказанного векторов для типичного (а), избегающего ошибок I рода (б) и избегающего ошибок II рода(в) алгоритмов

Precision и recall. Accuracy

Данные метрики рассчитываются после бинаризации предсказанных значений порогом . На рисунке 6 представлены типы объектов в зависимости от пар действительных и предсказанных значений.

		Предсказанный класс	
		1	0
Реальный класс	1	True Positive	False Negative
	0	False Positive	True Negative

Рисунок 1.6 – Типы объектов: True, False – верно и неверно предсказанный классобъекта соответственно; Positive, Negative – предсказанный класс объекта “1” (“Yes”) и “0” (“No”) соответственно

Наиболее простая и известная метрика – accuracy. Она показывает долю верно предсказанных примеров:

$$Accuracy = \frac{Tp + Tn}{Tp + Tn + Fp + Fn},$$

где $Tp - True\ Positive,\ Tn - True\ Negative,\ Fp - False\ Positive,\ Fn - False\ Negative$.

Precision и recall также широко распространены. Первая метрика показывает насколько верно предсказаны объекты, которым моделью была выставлена “1”, а вторая – точность предсказания примеров, в действительности относящихся к классу “1” (рисунок 1.7).

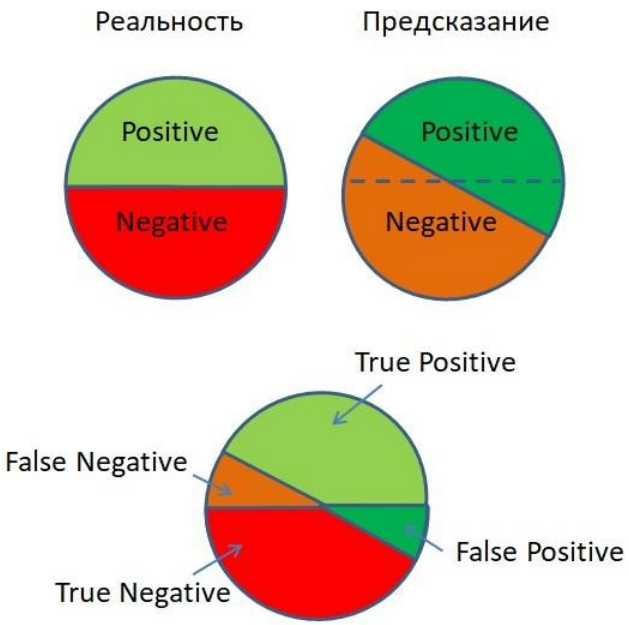


Рисунок 1.7 – Визуализация типов ошибок бинарной классификации

Расчёт данных метрик осуществляется по следующим формулам:

$$Precision = \frac{Tp}{Tp + Fp},$$

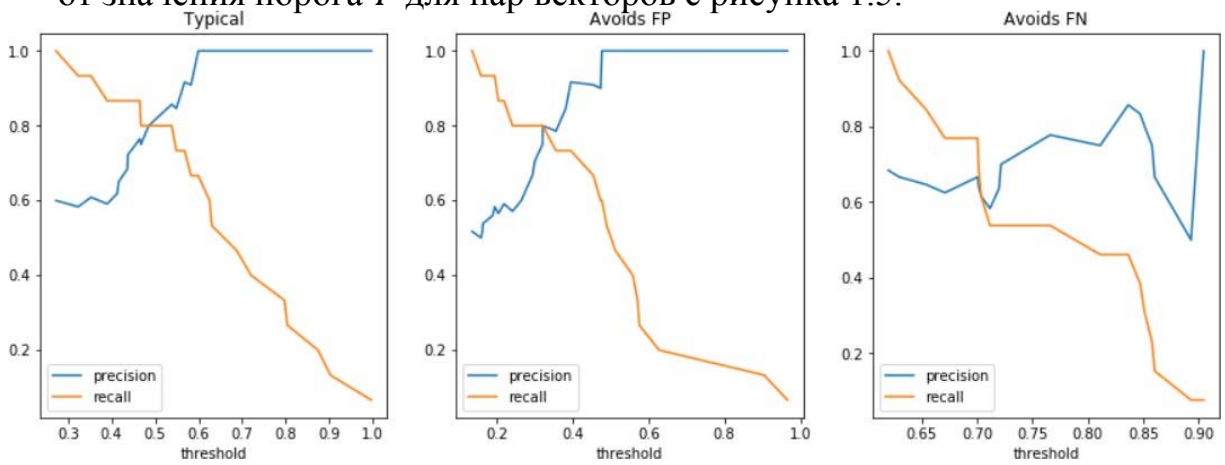
$$Recall = \frac{Tp}{Tp + Fn}.$$

С помощью всех трех метрик можно с легкостью определять случаи хорошо и плохо обученных алгоритмов классификации. К тому же, так как возможные значения лежат в интервале $[0, 1]$, то их легко интерпретировать.

Из данных метрик ничего нельзя узнать о самих значениях вероятностей объектов; можно определить только, какая их доля лежит не по ту сторону от порога T.

Метрика accuracy в равной мере штрафует алгоритм за наличие ошибок I и II родов. В то же время использование пары precision и recall позволяет четко устанавливать соотношение между родами ошибок: данные метрики используются для контроля Fp и Fn ошибок соответственно.

На рисунке 1.8 представлены метрики precision и recall в зависимости от значения порога T для пар векторов с рисунка 1.5.



а б в

Рисунок 1.8 – Precision и recall при различных значениях порога T для типичного (а), избегающего ошибок I рода (б) и избегающего ошибок II рода (в)
алгоритмов

По мере возрастания значения порога T мы получаем меньше Fp и больше Fn ошибок, согласно тому факту, что одна из кривых поднимается, а вторая падает вниз. Исходя из данной закономерности, можно выбрать оптимальный порог, при котором метрики precision и recall находятся в приемлемом интервале значений. Если такой порог не был найден, следует найти другой алгоритм для обучения.

Необходимо упомянуть, что приемлемые значения precision и recall определяются прикладным характером задачи. Например, в случае решения проблемы, имеется ли у пациента рассматриваемая болезнь (“0” – здоров, “1” – болен), Fn ошибки крайне нежелательны, отсюда значение метрики recall выставляется $\approx 0{,}9$, . Мы можем сказать пациенту, что он болен, и дальнейшей

диагностикой выявить ошибку, что намного лучше по сравнению с игнорированием реально имеющейся болезни.

F1-score

Очевидный недостаток пары precision-recall в том, что мы рассчитываем две метрики: при сравнении алгоритмов неясно, как использовать обе сразу. Решением данной проблемы является метрика $F1\text{-}score$:

$$F1\text{-}score = 2\frac{precision*recall}{precision+recall}$$

F1 метрика будет равна единице только тогда, когда $precision = 1$ и

$recall = 1$ (т.е. в идеальном алгоритме).

Довольно сложно ввести в заблуждение с помощью $F1\text{-}score$: если одна из составляющих близка к 1, а вторая показывает низкие значения (а такую ситуацию легко получить согласно рис. 1.8), то F1 метрика будет далека от идеального значения. Данную метрику трудно оптимизировать, так как для этого нужна как высокая точность, так и сбалансированность между родами ошибок.

Для пар векторов, представленных на рисунках 1.5а, 1.5б, 1.5в, значение

$F1\text{-}score$ при $T = 0{,}5$, составило 0,828, 0,636 и 0,765 соответственно. Значения метрики для второго и третьего примеров, где одно значение из пары precision- recall равнялось единице, оказались меньше по сравнению с первым сбалансированным случаем.

Описанные метрики легко интерпретировать, но при этом мы не берем вовнимание большую часть информации, получаемую с выхода модели. В некоторых задачах нужны вероятности в чистом виде (т.е. без их бинаризации).Например, при выставлении ставки на выигрыш футбольной команды нужно знать не класс, к которому будет отнесен объект, а вероятность его отнесения. Перед бинаризацией предсказаний может быть полезно рассмотреть вектор вероятностей на присутствие каких-либо закономерностей.

Логистическая функция ошибок (log_loss)

Метрика определяет среднее расхождение между вероятностямиотнесения объектов к конкретным классам и их действительными классами:

$$log\_loss = -\frac{1}{n}\sum_{i=1}^{n}[actual_i * \log(predicted_i) + (1 - actual_i) * \log(1 - predicted_i)],$$

где $actual_i$ – действительный класс i-го объекта, $predicted_i$ − вероятность отнесения i-го объекта к классу “1”, n – количество объектов. Функцию необходимо минимизировать.

Далее можно видеть значение функции ошибок для ранееиспользованных пар векторов.

Алгоритмы, разные по качеству (рис. 1.2):

Идеальный – 0,249

Типичный – 0,465

Неверно обученный – 1,527

Осторожный и рискующий алгоритмы (рис. 1.3 и 1.4):

Идеальный осторожный – 0,249

Идеальный рискующий – 0,171Типичный осторожный – 0,465Типичный рискующий – 0,614

Разные склонности алгоритмов к Fp и Fn ошибкам (рис. 5):Избегающий Fp ошибок – 0,585

Избегающий Fn ошибок – 0,589

Как и предыдущие метрики, *log_loss* может различать хорошо и плохо обученные алгоритмы, но при этом значения метрики трудно интерпретировать: она не может достичь нуля и не имеет ограничения сверху. Таким образом, даже для абсолютно точного алгоритма, если взглянуть на его значение логистической функции ошибок, невозможно будет сказать, что он идеален.

С другой стороны, метрика делает различия между осторожным и рискующим алгоритмами. Как видно по рис. 1.3б и 1.4б, число ошибочных примеров при пороге бинаризации $T = 0$, для типичной осторожной и рискующей моделей приблизительно равно, а в случае идеальных алгоритмов (рис. 1.3а, 1.4а) ошибок нет совсем. Однако рискующий алгоритм вносит больше веса в метрику *log_loss* за неверно подобранные предсказания по сравнению с осторожным, если модель является типичной, и меньше, если модель абсолютно точная.

Таким образом, *log_loss* чувствительна к вероятностям, близким как к 0 и 1, так и к 0,5.

Fp и Fn ошибки метрика выявить не может, но несложно перейти к болееобобщенной версии функции ошибок, где можно поднять штраф для того или иного рода ошибок. Для этого добавим комбинацию неотрицательных и дающих в сумме единицу коэффициентов при вероятностных членах функции. Например, если нужно больше штрафовать Fp ошибки:

$$weighted\_log\_loss(actual, predicted) =$$

$$= -\frac{1}{n}\sum_{i=1}^{n}[0{,}3 * actual_i * \log(predicted_i) + 0{,}7 * (1 - actual_i) * \log(1 - predicted_i)].$$

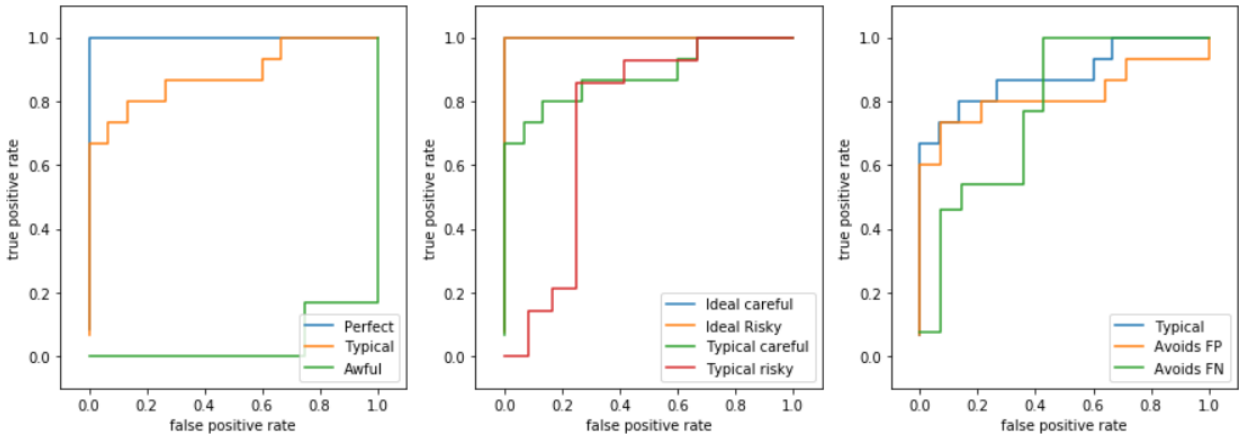
Если алгоритм выдает высокое значение вероятности, а объект вдействительности принадлежит к классу “0”, то первый член функции будет равен нулю, а второй будет рассчитан с учетом его большего веса.

ROC и AUC

При построении ROC-кривой (Receiver Operating Characteristic) подвергается варьированию порог бинаризации и рассчитываются некоторые величины, зависящие от количества Fp и Fn ошибок. Эти параметры подбираются таким образом, что в случае наличия порога для идеального разделения классов ROC-кривая будет проходить через определенную точку – левый верхний угол квадрата [0, 1] [0, 1]. В дополнение к этому, кривая всегда начинается в

левом нижнем и заканчивается в правом верхнем углу. Для того, чтобы охарактеризовать кривую численно, используется метрика AUC (Area Under the Curve) – площадь под ROC-кривой.

Далее визуализированы ROC-кривые для ранее использованных пар действительных и предсказанных векторов (рисунок 1.9).



а б в

Рисунок 1.9 – ROC-кривые для пар векторов, представленных на рисунках 1.2а, 1.3б и 1.4б, 1.5в

Значения AUC, соответствующие кривым на рис. 1.9, представлены ниже.Алгоритмы, разные по качеству (рис. 1.2):

Идеальный – 1,000

Типичный – 0,884

Неверно обученный – 0,042

Осторожный и рискующий алгоритмы (рис. 1.3 и 1.4):

Идеальный осторожный – 1,000

Идеальный рискующий – 1,000

Типичный осторожный – 0,884

Типичный рискующий – 0,738

Разные склонности алгоритмов к Fp и Fn ошибкам (рис. 1.5):

Избегающий Fp ошибок – 0,819

Избегающий Fn ошибок – 0,780

Чем больше объектов в выборке, тем более гладкими будут кривыми (хотя при приближении будет видно, что они все еще ступенчатые).

Как и ожидалось, кривые идеальных алгоритмов проходят через верхний левый угол. Также на рис. 1.9а желтой линией обозначена типичная ROC-кривая (в большинстве случаев кривая не может достичь данного угла).

Метрика AUC у рискующей модели значительно ниже по сравнению с осторожным алгоритмом, хотя в случае абсолютно точных моделей разницы между их ROC-кривыми и AUC метриками нет (рис. 1.9б). Таким образом, для идеального алгоритма нет смысла отдалять друг от друга точки, относящиеся к разным предсказанным классам.

При наличии большего числа Fp или Fn ошибок на кривых будет наблюдаться смещение (рис. 1.9в). Но по значению AUC его невозможно определить (в частном случае кривые могут быть симметричны относительно диагонали (0, 1) – (1, 0)).

После построения кривой удобно выбирать порог бинаризации, удовлетворяющий ограничениям по долям ошибок I или II рода. Определенное значение порога соответствует одной точке на ROC-кривой. Если мы хотим избежать Fp ошибок, то следует выбрать точку как можно ближе к левой стороне квадрата, если нежелательны Fn ошибки – ближе к верхней стороне. Все промежуточные точки отвечают за некоторые более сбалансированные соотношения между родами ошибок.

Часть из рассмотренных метрик качества классификации (например, log_loss) может быть перенесена на задачи, где объекты относятся к более чем 2классам. В случае, если затруднительно обобщить формулу метрики на несколько классов, такую задачу представляют в виде набора подзадач бинарной

классификации, а обобщенную метрику получают путем некоторого усреднения (micro- или macro-среднее) метрик подзадач.

На практике всегда полезно визуализировать векторы предсказанных алгоритмом значений с целью понять, какие ошибки совершает модель при различных порогах и как используемая метрика реагирует на них.

Задание 1. Скачайте данные и обучите 4 классификатора, чтобы предсказать поле "Activity" (биологический ответ молекулы) из набора данных "bioresponse.csv": мелкое дерево решений; глубокое дерево решений; случайный лес на мелких деревьях; случайный лес на глубоких деревьях.

Задание 2. Рассчитайте следующие метрики, чтобы проверить качество вашихмоделей: доля правильных ответов (*accuracy*); точность; полнота; $F1\text{-}score$; *log-loss*.

Задание 3. Постройте *precision-recall* и ROC-кривые для ваших моделей.

Задание 4. Обучите классификатор, который избегает ошибок второго рода ирассчитайте для него метрики качества.

Контрольные вопросы

1. Какую метрику нужно оптимизировать, чтобы настроить алгоритм избегать ошибок первого рода (FP)?

2. На мелких или на глубоких деревьях лучше строить случайный лес с точки зрения качества алгоритма классификации?

3. Проанализируйте качество алгоритма классификации для алгоритма случайного леса на 5, 10, 25 и 50 деревьях. Как меняется точность решения? Почему?

Литература

1. Евгений Соколов. Семинары по метрическим методам классификации. http://www.machinelearning.ru/wiki/images/9/9a/Sem1_knn.pdf

2. К. В. Воронцов. Метрические методы классификации и регрессии. Лекция. http://www.machinelearning.ru/wiki/images/c/c3/Voron-ML-Metric-slides.pdf

Лабораторная работа № 2

Предобработка данных. Отбор признаков

Цель работы: получение навыков предобработки данных, необходимых для качественной настройки моделей машинного обучения.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: изучение методик предобработки данных, методов отбора

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

При использовании данных различной природы в машинном обучении зачастую приходится сталкиваться с зашумлёнными данными или данными, в которых встречаются выбросы и взаимозависимые признаки. Для эффективной настройки моделей машинного обучения необходимо заниматьсяпредварительной обработкой данных (data preprocessing).

Существует несколько методик предобработки данных.

Масштабирование и нормализация (normalization, scaling). Для уменьшения влияния выбросов используют нормировку данных, когда количественные признаки отображают, например, на отрезок [0,1] или [-1,1].

Центрирование. Также для уменьшения влияния выбросов используют нормировку, делая матожидание равным нулю.

One-hot-encoding. Данный приём используется, когда признаки являются категориальными, то есть принимают значения, между которыми нельзя установить отношения порядка. Мы можем закодировать каждое из таких значений бинарным вектором, длина которого будет равна количеству возможных значений данного признака, и на всех его позициях будут стоять нули за исключением индекса, соответствующего текущему значению. Данная операция порой значительно увеличивает размерность задачи, но позволяетэффективно включать в модель категориальные признаки.

Отбор признаков. Некоторые признаки иногда не дают никакого вклада

вулучшение качества работы алгоритма, а порой бывают даже вредными. Поэтому используют фильтрацию, отбрасывая ненужные признаки, с учётом некоторого критерия (например, корреляции Пирсона между признаками и целевой переменной).

Также для эффективной настройки моделей машинного обучения и борьбы с переобучением используют технику кросс-валидации (cross-validation). Кросс-валидация - процедура эмпирической оценки обобщающей способности алгоритмов. С помощью кросс-валидации эмулируется наличие тестовой выборки посредством последовательного разделения всей выборки данных на тренировочную и тестовую, при этом последняя не участвует вобучении, но для неё известны правильные ответы.

Данные содержат некоторые атрибуты, которые могут быть либо излишними – сильно коррелирующими с другими, – либо незначимыми, т.е. слабо влияющими на целевую переменную, а потому могут быть удалены без существенной потери информации. Отбор признаков, особенно при большом их количестве в данных, нужен, потому что, во-первых, если признаков очень много (десятки или сотни), то увеличивается время обучения моделей; во-вторых, с увеличением количества признаков часто падает точность предсказания, особенно, если в данных много признаков, слабо коррелирующих с целевой переменной.

Есть три категории методов отбора: фильтрация (filter methods), оборачивание (wrapper methods) и встроенные методы (embedded methods).

Фильтрация параметров

Можно применить фильтрацию параметров по критерию увеличения информации (informaition gain), вычисляемой следующим образом:

$$IG\ (Y|X) = H(Y) - H(Y|X)$$

разница между значением обычной энтропии признака Y и относительной энтропией (specific conditional entropy), для которой энтропия $H(Y)$ рассчитывается только для тех записей, для которых $X=x_i$. Т.е. это мера того, насколько

более упорядоченной становится для нас переменная *Y*, если мы знаем значения *X*, или, говоря проще, существует ли корреляция между значениями *X* и *Y*, и насколько она велика.

$$H(Y) = -\sum_{y_i \in Y} p(y_i) \cdot \log_2 p(y_i)$$

$$H(Y|X) = \sum_{x_i \in X} p(x_i) \cdot H(Y|X = x_i)$$

,

где $p(x_i)$ – вероятность того, что переменная *X* примет значение x_i. В наших условиях эта вероятность считается как количество записей (примеров), в которых X = x_i, разделенное на общее количество записей. Чем больше параметр IG – тем сильнее корреляция. Таким образом, можно вычислить information gain для всех признаков и выкинуть те, которые слабо влияют на целевую переменную.

На рисунке 2.1 показан пример рассчитанных корреляций на данных для классификации мобильных телефонов по ценовым категориям. Можно видеть, что наибольшее влияние на целевую функцию (price_range) оказывает размер оперативной памяти (ram), в то время как вклад признака наличия двух симкарт (dual_sim) незначителен.

На том же наборе данных можно определить 10 атрибутов, наиболеезначимых для классификации (см. рисунок 2.2). Видно, что, как и в первом случае, вклад переменной ram в определение ценовой категории наибольший.

У методов фильтрации низкая стоимость вычислений, которая зависит линейно от общего количества признаков. Они значительно быстрее и wrapping, и embedded методов и хорошо работают даже тогда, когда число признаков превышает количество примеров в тренировочном наборе данных. Недостаток в том, что они рассматривают каждый признак изолированно, а наиболее коррелирующие признаки не всегда составляют подмножество, на котором точность предсказания будет наивысшей.

Wrapping.

Суть этой категории методов в том, что классификатор запускается наразных подмножествах признаков исходного тренировочного набора данных.

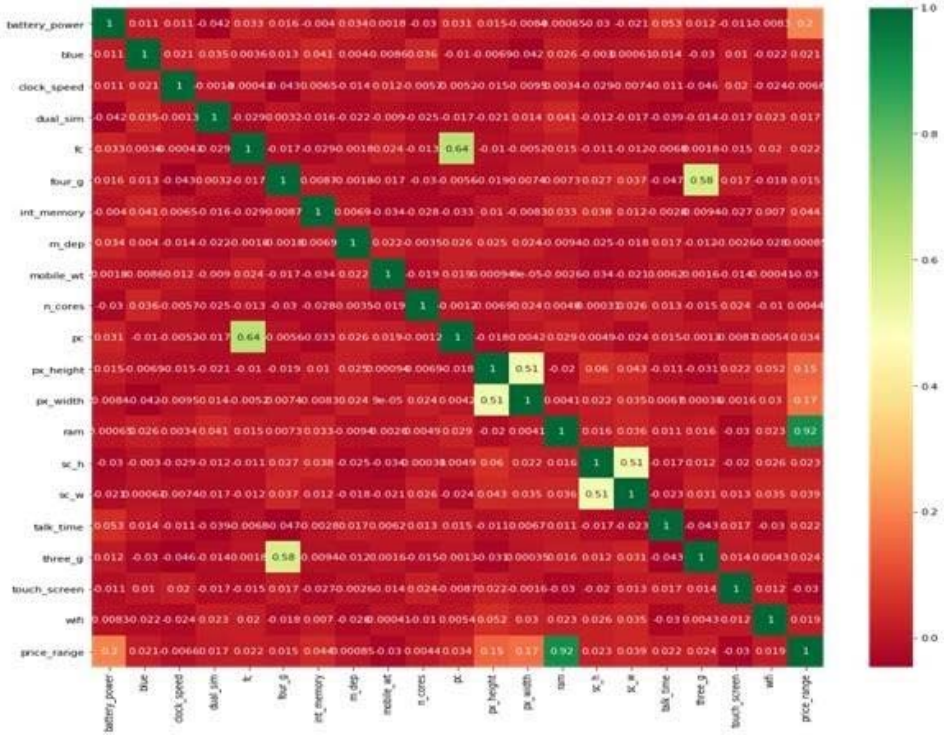


Рисунок 2.1 – Корреляция признаков между собой и целевой переменной После этого выбирается подмножество признаков с наилучшими

параметрами на обучающей выборке, а затем он тестируется на тестовом сете.

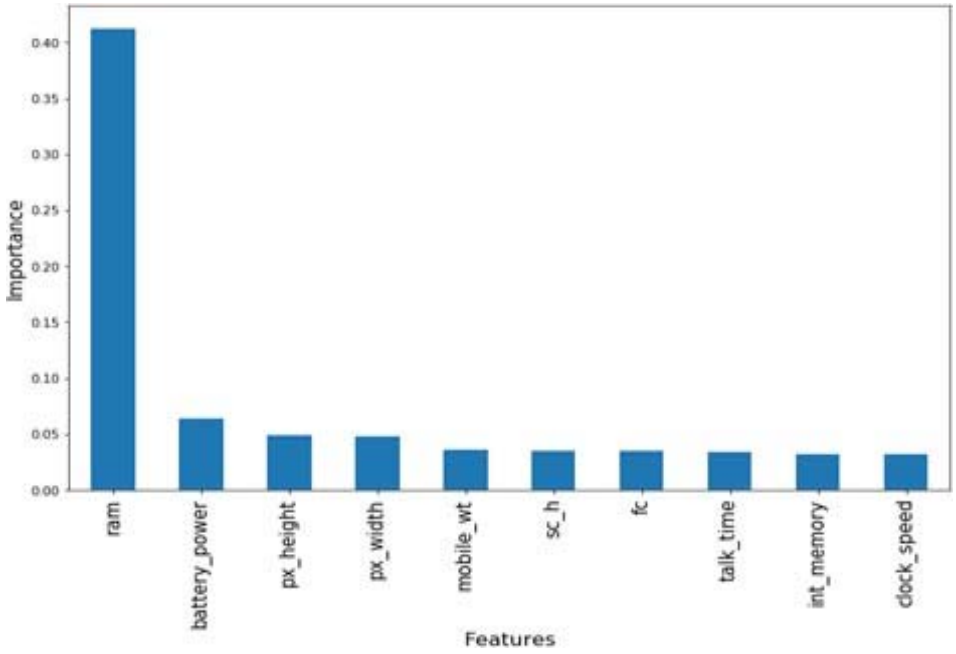


Рисунок 2.2 – Оценка значимости признаков по критерию $\chi 2$

Существует два подхода в этом классе методов: методы включения (forward selection) и исключения (backwards selection) признаков. Первые стартуют с пустого подмножества, куда постепенно добавляются разные признаки (для выбора на каждом шаге оптимального добавления). Во втором случае метод стартует с подмножества, равного исходному множеству признаков, и из него постепенно удаляются признаки, с итерационным переобучением классификатора.

Встроенные методы (embedded)

Эти методы позволяют не разделять отбор признаков и обучение классификатора, а производят отбор внутри процесса расчета модели. Эти алгоритмы требуют меньше вычислений, чем wrapper methods (хотя и больше, чем методы фильтрации). Основным методом из этой категории является регуляризация.

а) L2 – метод регуляризации Тихонова (ridge regression)

В случае построения модели линейной регрессии, если в тестовом наборе данных задана матрица объектов-признаков A и вектор целевой переменной b, то мы ищем решение в виде $Ax=b$. В процессе работы алгоритма минимизируется следующее выражение:

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{i,j}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 ,$$

где первое слагаемое — это среднеквадратичная ошибка модели, а второе — регуляризирующий оператор (сумма квадратов всех коэффициентов, умноженная на λ). В процессе работы алгоритма размеры коэффициентов будут пропорциональны важности соответствующих признаков, а для тех признаков, которые дают наименьший вклад в устранение ошибки, будут близки к нулю. Параметр λ позволяет настраивать вклад регуляризирующего оператора в общую сумму. С его помощью мы можем указать приоритет: точность модели или минимальное количество используемых переменных.

б) L1 – метод регуляризации LASSO (Least Absolute Shrinkage and Selection Operator)

Метод *L1* аналогичен предыдущему во всем, кроме отличия в регуляризирующем операторе. Он представляет собой не сумму квадратов, а сумму модулей коэффициентов:

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{i,j}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|,$$

Несмотря на внешнюю схожесть, методы отличаются. Если в Ridge по мере роста λ все коэффициенты постепенно получают значения, близкие к нулевым, то в LASSO с ростом λ всё больше коэффициентов становятся нулевыми и совсем перестают вносить вклад в модель. Таким образом, реализуется отбор признаков.

Эти методы являются отличной альтернативой пошаговой регрессии и wrapper-методам, когда необходимо работать с набором данным с большим количеством признаков.

Задания для выполнения лабораторной работы

Задание. Скачайте данные для исследования:

1. Реализуйте функции one-hot-encoding и softmax средствами базового Python.

2. Разделите переменные на численные и категориальные, масштабировать и нормировать данные.

3. Реализуйте модель логистической регрессии средствами базового Python для решения задачи бинарной классификации для полного набора признаков из предобработанных данных и для непредобработанных данных, а также только для количественных признаков.

4. Сделайте вывод об изменении качества работы модели в зависимости

от применения предобработки данных и объёма признаков, на которых обучалась модель.

Контрольные вопросы

1. Для чего служит преобразование one-hot-encoding? Всегда ли использование этого метода будет повышать качество работы модели машиного обучения?

2. Почему регуляризатор L1 (Lasso) обнуляет веса при линейно зависимыхпризнаках?

Литература

1. https://towardsdatascience.com/feature-engineering-for-machine-learning- 3a5e293a5114

2. К. В. Воронцов. Отбор признаков. Лекция. http://www.machinelearning.ru/wiki/images/archive/4/4f/20111004204412%2 1Voron-ML-Modeling-slides.pdf

Лабораторная работа 3

Функции ошибок в машинном обучении

Цель: получение знаний и критериев применимости основных используемых в современном машинном обучении функций ошибок (функций потерь).

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: критерии применимости основных используемых в современном машинном обучении функций ошибок (функций потерь)

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

Функция потерь – функционал, оценивающий величину расхождения между истинным значением оцениваемого параметра и модельной оценкой этого параметра.

Задачи регрессии

Выбирая функцию потерь для задач регрессии, следует решить, какое именно свойство условного распределения мы хотим восстановить. Наиболее частые варианты:

$L(y,f) = (y-f)^2$ - Gaussian loss (*L2*), самый часто используемый и простой вариант, если нет никакой дополнительной информации или требований к устойчивости модели.

$L(y,f) = |y-f|$ – Laplacian loss (*L1*); в некоторых задачах эта функция потерьпредпочтительнее, так как она не так сильно штрафует большие отклонения, нежели квадратичная функция.

$$L(y,f) = \begin{cases} (1-\alpha)|y-f| & \text{при } y-f \le 0 \\ \alpha|y-f| & \text{при } y-f > 0 \end{cases}$$

- Quantile loss (*Lq*), функция асим-

метрична и больше штрафует наблюдения, оказывающиеся по нужную сторону квантили.

Графики перечисленных функций приведены на рисунке 3.1.

Задачи классификации

При классификации из-за принципиально другой природы распределенияцелевой переменной оптимизируются не сами метки классов, а их *log*-правдоподобие. Наиболее известные варианты таких классификационныхфункций потерь изображены на рисунке 3.1.

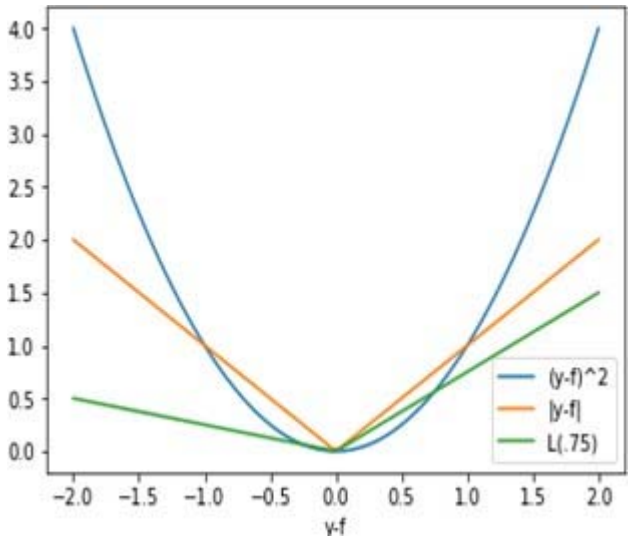


Рисунок 3.1 – Функции потерь для задач регрессии

Математические выражения для расчёта этих функций следующие:

$L(y, f) = \ln(1 + \exp(-yf))$ - Logistic loss (Bernoulli loss); штрафуются даже корректно предсказанные метки классов, но, оптимизируя эту функцию потерь, можно улучшать классификатор, даже если все наблюдения предсказаны верно. Это самая стандартная и часто используемая функция потерь в бинарной классификации.

$L(y, f) = \exp(-yf)$ - Adaboo.st loss; имеет более жесткий экспоненциальный штраф на ошибки классификации и используется реже.

Для использования в задачах классификации применима также функция

кросс-энтропии, которая определяет меру расхождения между двумя вероятностными распределениями. Если кросс-энтропия велика, разница между двумя распределениями велика, а если кросс-энтропия мала, распределения похожи друг на друга:

$$H(P,Q) = -\sum P(x) \cdot \log_2 Q(x),$$

где P – распределение истинных ответов, а Q – распределение вероятностей прогнозов модели.

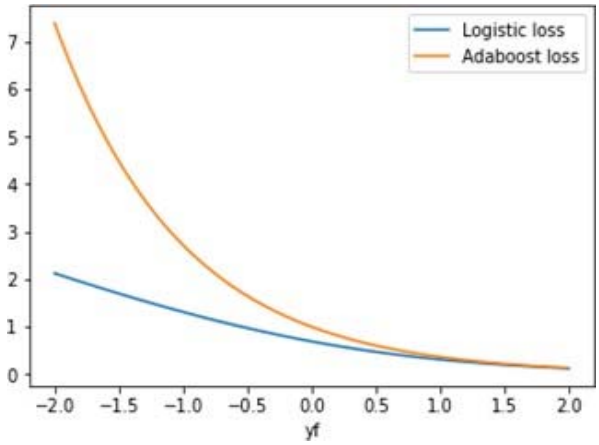


Рисунок 3.2 – Функции Logistic loss и Adaboost loss

В случае бинарной классификации формула для расчёта кросс-энтропии имеет следующий вид:

$$L = -\mathrm{y} \cdot \log_2 p + (1-y) \cdot \log_2 (1-p),$$

где y – двоичный индикатор (0 или 1) того, является ли метка класса правильной классификацией для текущего наблюдения, p – прогнозируемая вероятность класса, определённая моделью классификации.

При бинарной классификации каждая предсказанная вероятность сравнивается с фактическим значением класса (0 или 1) и вычисляется оценка, которая штрафует вероятность пропорционально величине отклонения от ожидаемого значения. Конфигурация выходного уровня модели представляет собой один узел с сигмоидальной функцией активации:

$$f(p_i) = \frac{1}{1 + \exp(-p_i)}$$

Чтобы классифицировать объект как принадлежащий одному из нескольких классов, задача формулируется как предсказание вероятности того, что пример принадлежит каждому классу. В случае, когда классов много (*M>2*)берется сумма значений логарифмических функций потерь для каждого прогноза наблюдаемых классов – «categorical cross-entropy»:

$$CE = -\sum_{c=1}^{M} y_{o,c} \cdot \log_2 p_{o,c}$$

.

Для расчёта взвешенного вектора вероятностей отнесения объекта к конкретным классам используется функция активации «softmax» — обобщение логистической функции для многомерного случая.

$$f(p_i) = \frac{\exp(p_i)}{\sum_{j=1}^{M} \exp(p_j)}$$

.

Когда каждый из объектов должен классифицироваться однозначно, что чаще всего и требуется, можно отбросить слагаемые, которые являются нулевыми из-за значений целевых индикаторов *y*. Тогда:

$$CE = -\log_2 \frac{\exp(p_i)}{\sum_{j=1}^{M} \exp(p_j)},$$

где p_i – результат оценки принадлежности к соответствующему классу.

Задание 1. Скачайте данные, используя базу любых открытых дата сетов:

1. Реализуйте модель логистической регрессии со следующими функциями потерь:

а) Logistic loss б) Adaboost loss

в) binary crossentropy

2. Визуализируйте кривые обучения модели бинарной классфикации в виде динамики изменения каждой из функций ошибок п.2 на тренировочной

и тестовой выборках.

3. Сравните качество классификации по метрике accuracy в каждом из трёхмодификаций алгоритма.

Контрольные вопросы

1. Какую функцию потерь нужно применять в задаче регрессии, если выхотите, чтобы модель больше штрафовала за выбросы в данных?

2. Как функция кросс-энтропии связана с дивергенцией Куль-бака-Лейблера?

Литература

1. Основные функции потерь и примеры их реализации на Python: https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/

2. Алгоритм градиентного бустинга с разбором функций потерь для регрессии и классификации: https://habr.com/ru/company/ods/blog/327250/#3-funkcii-poter

Лабораторная работа 4

Алгоритмы кластеризации

Цель: является получение навыков реализации классических алгоритмов кластеризации k-means и иерархической кластеризации.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на практическом занятии: методы классических алгоритмов кластеризации k-means и иерархической кластеризации

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

Постановка задачи кластеризации (или обучения без учителя) формулируется следующим образом. Пусть имеется обучающая выборка $X^l = \{x_1, \dots, x_l\} \subset X$ и функция расстояния между объектами ρ(x, x′), введённая в соответствующем метрическом пространстве размерности l. Необходимо разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике ρ, а объекты разных кластеров существенно отличались. При этом каждому объекту $x_i \in X^l$ приписывается метка (номер) кластера y_i.

Алгоритм кластеризации – это функция a X Y, которая каждому объекту $x \in$ X ставит в соответствие метку кластера $y \in Y$. В ряде случаев множество меток Y известно заранее, однако чаще ставится задача определить оптимальное число кластеров с точки зрения того или иного критерия качества кластеризации. В качестве метрики качества кластеризации можно взять, например, среднее (минимальное, максимальное) расстояние между объектамив разных кластерах, либо среднее взвешенное (с весами, взятыми пропорционально, например, размерам кластеров).

Цели кластеризации:

а) Классификация объектов.

Попытка понять зависимости между объектами путем выявления их

кластерной структуры. Разбиение выборки на группы схожих объектов упрощает дальнейшую обработку данных и принятие решений, позволяет применить к каждому кластеру свой метод анализа (стратегия «разделяй и властвуй»). В данном случае стремятся уменьшить число кластеров для выявления наиболее общих закономерностей;

б) Сжатие данных.

Можно сократить размер исходной выборки, взяв один или несколько наиболее типичных представителей каждого кластера (например, вблизи центра масс). Здесь важно точно очертить границы каждого кластера, при этом их количество не является важным критерием.

в) Обнаружение выбросов.

Нахождение таких объектов в данных, которые выделяются из общей массы и не подходят ни одному кластеру. Обнаруженные объекты в дальнейшем обрабатывают отдельно.

Наиболее популярные алгоритмы кластеризации:

а) Алгоритм *k*-средних и его модификации (*k*-means);б) Иерархические алгоритмы кластеризации;

в) Вероятностные алгоритмы кластеризации (например, EM- алгоритм);

г) Сдвиг среднего значения (англ. MeanShift);

д) Пространственная кластеризация на основе анализа плотности данных (англ. Density-based spatial clustering of applications with noise, алгоритм DBSCAN).

Рассмотрим подробнее классический алгоритм *k*-means и класс иерархических алгоритмов.

Алгоритм k-means.

Алгоритм кластеризация с помощью алгоритма *k*-средних следующий:

1. Выбрать начальное приближение Y центров всех кластеров: $y \in Y: \mu_y$

2. Отнести каждый объект x_i к одному из кластеров, исходя из минимума метрики расстояния $p(x_i, \mu_y)$:

do:

$$y_i := arg\min_{y \in Y} p(x_i, \mu_y), i = 1, \ldots, l$$

3. Вычислить новое положение центров каждого из кластеров, пока метки y_i не перестанут изменяться.

$$\mu_{yj} = \frac{\sum_{i=1}^{l}[y_i = y]f_i(x_i)}{\sum_{i=1}^{l}[y_i = y]}, y \in Y, j = 1, \ldots, n$$

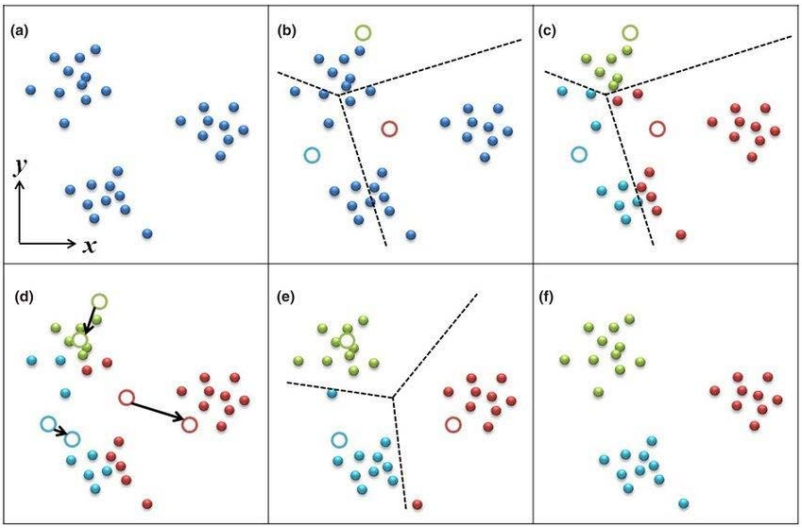


Рисунок 4.1 – Схематическая иллюстрация выполнения последовательных итерацийалгоритма *k*-means на двумерныхданных с разделением на 3 кластера

Из минусов данного алгоритма можно отметить крайнюю чувствительность к выбору начальных приближений центров. Случайная инициализация центров на шаге 1 может приводить к плохой работе алгоритма.Также *k*-means может показывать плохие результаты и в том случае, если изначально будет неверно угадано число кластеров. Стандартная рекомендациязаключается в проведении кластеризации при различных значениях *k* и выборе того варианта, при котором достигается резкое улучшение качества кластеризации по заданному функционалу (например, межкластерному расстоянию). Если перед началом работы точно неизвестно количество кластеров, можно использовать алгоритмы, которые самостоятельно определят их количество (например, DBSCAN).

Иерархические алгоритмы кластеризации

Алгоритмы иерархической кластеризации подразделяются на два основных типа: восходящие и нисходящие алгоритмы. Нисходящие алгоритмы работают по принципу «сверху-вниз»: вначале все объекты помещаются в один кластер, который затем разбивается на все более мелкие кластеры. Более распространены восходящие алгоритмы, которые в начале работы помещают каждый объект в отдельный кластер, а затем объединяют кластеры во все болеекрупные, пока все объекты выборки не будут содержаться в одном кластере. Таким образом строится система вложенных разбиений. Результаты таких алгоритмов обычно представляют в виде дерева – дендрограммы. Классическийпример такого дерева – классификация животных и растений.

К недостатку иерархических алгоритмов можно отнести систему полных разбиений, которая может являться излишней в контексте решаемой задачи.

Разберём восходящий алгоритм кластеризации. Вначале каждый объект считается отдельным кластером. Для одноэлементных кластеров естественным образом определяется функция расстояния $R(\{x\},\{y\})=\rho(x, y)$. Затем запускается процесс слияний. На каждой итерации вместо пары самых близких кластеров U и V образуется новый кластер $W = U \cup V$. Расстояние от нового кластера W до любого другого кластера S вычисляется по расстояниям R(U,V), R(U, S) и R(V, S), которые к этому моменту уже известны:

$$R(U \cup V, S) = \alpha U R(U, S) + \alpha V R(V, S) + \beta R(U,V) + \gamma |R(U, S) - R(V, S)|,$$

где αU, αU, β, γ — числовые параметры. Эта универсальная формула для расчёта расстояний между кластерами, которая была предложена Лансом и Уильямсом в 1967 году. Самыми распространёнными её вариантами являются метод ближайшего и метод дальнего соседа, которые вычисляют расстояние между кластерами как расстояние между наименее (наиболее) удаленными друг от друга их членами.

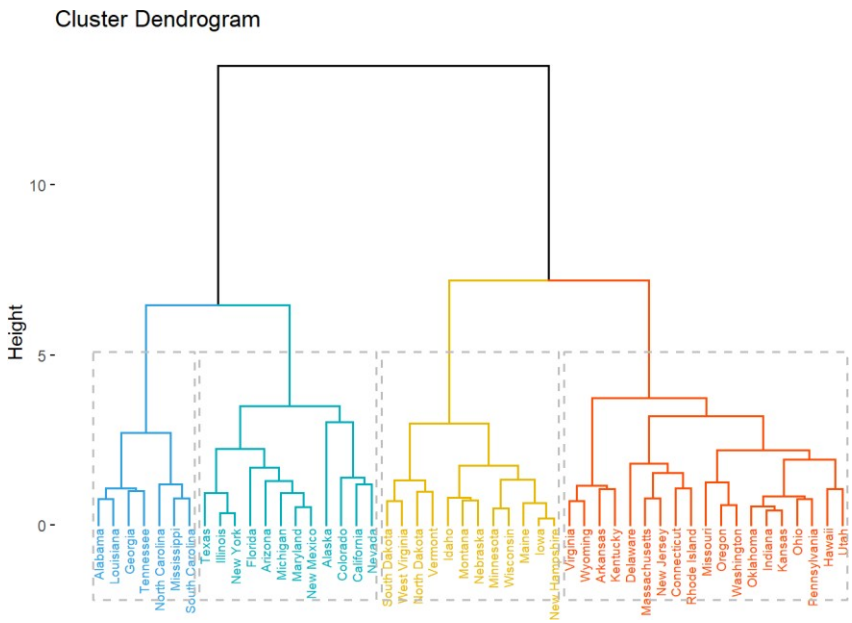


Рисунок 4.2 – Пример дендрограммы с последовательным объединением кластеров

Задание 1. Скачайте данные (MNIST): http://yann.lecun.com/exdb/mnist/

Задание 2. Реализуйте в виде набора функций алгоритм *k*-means и алгоритм иерархической кластеризации (например, с использованием функции *linkage* модуля *scipy.cluster*) для разделения набора данных без меток на кластеры.

Задание 3. Постройте кривую зависимости интеркластерного расстояния от числа кластеров для алгоритма иерархической кластеризации,выбрать оптимальный порог разделения.

Задание 4. Сравните результаты двух выбранных алгоритмов по выбранной метрики оценки качества кластеризации.

Контрольные вопросы

1. Каким образом следует выбирать порог разделения для иерархической кластеризации?

2. Как, по Вашему мнению, можно улучшить инициализацию центров кластеров в алгоритме k-means с целью нахождения наиболее оптимальных начальных приближений?

3. Реализуйте алгоритм DBSCAN на данных из лабораторной работы и сравните его качество с качеством работы алгоритма k-means и метода linkage.

Литература

1. Документация модуля **sklearn.cluster**. библиотеки sklearn: https://scikit-learn.org/stable/modules/clustering.html

2. Обзор методов обучения без учителя: https://habr.com/ru/company/ods/blog/325654/

3. Lance G. N., Willams W. T. A general theory of classification sortingstrategies. Hierarchical systems // Comp. J. – 1967. – v. 9. – Pp. 373–380.

Лабораторная работа № 5

Введение в обработку естественного языка

Цель: получение студентом навыков реализации базовых методов обработки естественного языка, включая предобработку текста, формирование «мешка слов» («bag-of-words»), выделение стоп-слов и наиболее важных слов в документе, создание тематических моделей.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на практическом занятии: методы обработки естественного языка, включая предобработку текста, формирование «мешка слов» («bag-of-words»), выделение стоп-слов и наиболее важных слов в документе, создание тематических моделей.

Формируемые компетенции: ПК-2, ПК-3, ПК-8: ПК-9: ПК-12.

Теоретическая часть

Задачи обработки естественного языка – одни из самых востребованных всовременном машинном обучении. Они включают в себя такие области, как машинный перевод, аннотирование текстов, классификация документов, генерация текста, text-to-image и text-to-video задачи, построение чат-ботов и диалоговых систем и многие другие.

Решение любой задачи в сфере NLP (natural language processing) начинается с предобработки текста, которая обычно включает в себя:

а) удаление всех не относящиеся к естественному языку символов из текста: @, #, & и т.д. (если это не противоречит цели исследования);

б) токенизиция текста, т.е., разделение его на отдельные слова: слово = «токен»;

в) удаление стоп-слов, встречающихся во всех без исключения текстах и не несущих смысловой нагрузки для решения текущей задачи;

г) приведение текста к нижнему регистру, чтобы модель распознавала такие слова, как, например, «привет» и «Привет», одинаково;

д) стэмминг (обрезка слова до его основания) и лемматизация текста

(приведение слов к единой форме, например, («красивая», «красивый», «красивое») → «красивый»).

Чтобы использовать методы машинного обучения на текстовых документах, нужно перевести текстовое содержимое (слова на естественном языке) в числовой вектор признаков. Наиболее интуитивно понятный способ сделать описанное выше преобразование — это представить текст в виде набора слов, после чего приписать каждому слову в тексте уникальный целочисленный индекс, соответствующий частоте его появления в документах обучающей выборки. Для этого в каждом документе *i* следует посчитать количество употреблений каждого слова *w* и сохранить это число в ячейке *X[i, j]*. Это будет соответствовать значению признака *j*, где *j* — это индекс слова *w* в словаре.

Такое преобразование текста в матрицу частот употреблений слов носит название «мешка слов» (или «bag of words»). Оно подразумевает, что вероятность появления слова в разных документах одинакова, и строит матрицуобъекты-признаки исходя из предположения, что количество признаков соответствует количеству уникальных слов в корпусе документов. «Мешок слов» чаще всего является высокоразмерным разреженным набором данных (с большим количеством нулей).

Рисунок 5.1 – Пример построения матрицы частот упоминаний уни-

D1 - “I am feeling very happy today”
D2 - “I am not well today”
D3 - “I wish I could go to play”

	I	am	feeling	very	happy	today	not	well	wish	could	go	to	play
D1	1	1	1	1	1	1	0	0	0	0	0	0	0
D2	1	1	0	0	0	1	1	1	0	0	0	0	0
D3	2	0	0	0	0	0	0	0	1	1	1	1	1

кальных слов для корпуса документов из трёх предложений (источник изображения:deeplearning.ai by Andrew Ng)

Однако, даже если документы посвящены одной теме, в относительно длинных документах среднее количество словоупотреблений будет выше, чем

в коротких, что может приводить к некорректной настройке моделей машинного обучения. Чтобы избежать потенциальных несоответствий, применятеся подход, называемый *TF-IDF* – «term frequency – inverse document frequency», который позволяет оценить «важность» слова для данного документа. Этот подход позволяет снизить влияние низкоинформативных слов и, наоборот, повысить «вес» важных с точки зрения оценки принадлежности документа к определённой тематике. Для каждого слова вычисляется следующая величина:

$$tf - idf\ (t, d, D) = tf(t, d) \times idf(t, D),$$

$$tf(t, d) = \frac{n_t}{\sum_k n_k},$$

$$idf(t, D) = log\ \frac{|D|}{|\{d_i \in D | t \in d_i\}|},$$

где n_t - частота слова (токена) *t* в документе *d,* | *D* | - количество документов вколлекции; $|\{d_i \in D | t \in d_i\}|$ - количество документов в коллекции, в которых встречается слово t.

Большое значение *TF-IDF* будут получать слова с высокой частотой внутри конкретного документа и с низкой частотой использования в других документах. Заполняя матрицу объекты-признаки значениями *TF-IDF,* можно эффективнее выделять важные «признаки» (слова) и проводить более качественную настройку моделей машинного обучения.

Более продвинутыми с точки зрения выделения семантики различных слов являются современные решения, полученные в результате обучения на больших корпусах документов. К ним относится, в частности, инструмент word2vec, разработанный в 2013 году компанией Google и получивший широкое распространение. Word2vec вычисляет векторное представление слов, обучаясь на входных текстах. Это векторное представление основывается на контекстной близости: слова, встречающиеся в тексте рядом с одними и теми же словами (а, следовательно, имеющие схожий смысл), будут иметь близкие координаты в многомерном пространстве (норма расстояния между их векторными представлениями будет мала).

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)
Gender	−1	1	-0.95	0.97
Royal	0.01	0.02	0.93	0.95
Age	0.03	0.02	0.70	0.69
Food	0.09	0.01	0.02	0.01

Рисунок 5.2 – Пример формирования векторного представления слов в методеword2vec (источник изображения: deeplearning.ai by Andrew Ng)

Сформированные таким образом вектора позволяют вычислять «семантическое расстояние» между словами: например, слова «король» и «королева» будут находиться в этом представлении близко друг к другу, аслова «король» и «яблоко» - далеко.

Задания к лабораторной работе

Задание 1. Скачайте любой датасет из открытых источников, содержащий текстовую информацию

Задание 2. Реализуйте пайплайн обработки выбранного текста на английском языке, включая всю необходимую предварительную обработку текста, включая приведение слов к нижнему регистру, удаление стоп-слов, цифр/неалфавитных символов, знаков пунктуации.

Задание 3. Разделите текст на главы и в каждой главе отобрать Топ-20 слов с помощью алгоритма TF-IDF.

Задание 4. Реализуйте LDA алгоритм и сравнить результаты с полученными ранее с помощью TF-IDF. Сделайте выводы о применимостиреализованных подходов.

Контрольные вопросы

1. Какое слово имеет максимальное значение метрики TF-IDF длядвенадцатой главы под названием «Alice’s Evidence»?

2. Как бы вы назвали главы книги «Alice in Wonderland», основываясь нанайденных для каждой главы важных словах?

Литература

1. NLP с примерами на Python: https://habr.com/ru/company/Voximplant/blog/446738/
2. Word2vec, презентация: http://www.machinelearning.ru/wiki/images/b/b3/Word2Vec.pdf

Лабораторная работа № 6

Методы оптимизации в глубоком обучении

Цель: получение навыков реализации современных алгоритмов оптимизации и модификаций алгоритма градиентного спуска, широко использующегося для обучения глубоких нейронных сетей.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на практическом занятии: алгоритмы оптимизации и модификации алгоритма градиентного спуска

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

В общем случае задачу оптимизации можно записать в виде:

$$minimize_x\, f(x) \text{ при } \begin{cases} g_i(x) \le 0, & i = 1, \dots, m \\ h_j(x) = 0, & j = 1, \dots, p \end{cases},$$

где $f: \mathbb{R}^n \to \mathbb{R}$ – целевая функция, минимизация которой осуществляется путемпоиска значений -мерного вектора x;

$g_i(x) \le 0$ – ограничения в виде неравенств;

$h_j(x) = 0$ – ограничения в виде равенств;

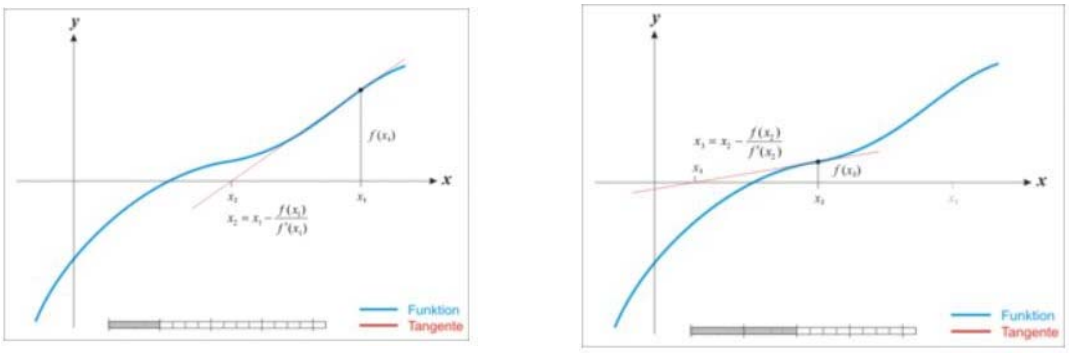
$m \ge 0$, $p \ge 0$.

В рамках лабораторной работы мы сосредоточимся на рассмотрении дифференциальных методов оптимизации заданной функции нескольких переменных и вариациях метода градиентного спуска.

Метод Ньютона

Метод Ньютона – метод поиска корней уравнения, использующий линейную аппроксимацию. Предполагается, что точка x_n – некоторое приближенное решение уравнения $f(x) = 0$. В точке производится расчет линейного приближения функции $f(x)$, после чего в качестве нового приближенного решения x_{n+1} берется пересечение графика аппроксимации с осью абсцисс

(рисунок 6.1).



а б

Рисунок 6.1 – Принцип работы метода Ньютона: (а), (б) – первая и второйитерации метода соответственно

Например, методом Ньютона после 6 итераций было найдено решение x =1, уравнения $x - x^2 - 1 = 0$ при $x = 1$ (рисунок 6.2).

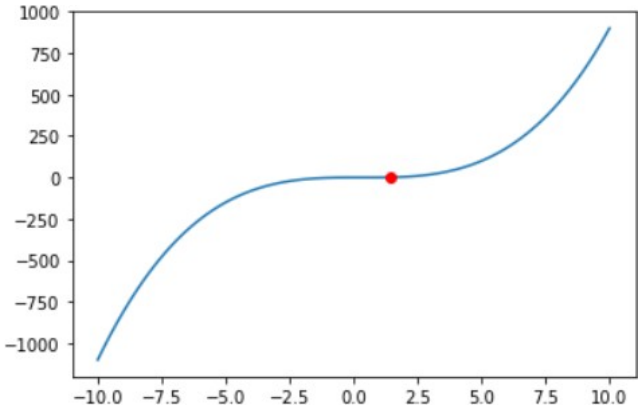


Рисунок 6.2 – Функция $f(x) = x - x^2 - 1$ (синяя линия) и решение уравнения

$f(x) = 0$ (красная точка), найденное методом Ньютона

Пакет scipy.optimize содержит в себе большое количество часто применяемых на практике алгоритмов оптимизации. Данный модуль предоставляет:

а) возможность решения задач безусловной и условной минимизации скалярных функций нескольких переменных через метод minimize(), использующий такие алгоритмы, как BFGS (алгоритм Бройдена- Флетчера-Гольдфарба-Шанно), Nelder-Mead simplex (симплекс-метод Нелдера-Мида),

Newton Conjugate Gradient (метод сопряженных градиентов) и другие;

б) реализацию методов глобальной оптимизации грубой оценки (например,anneal() – алгоритм Метрополиса, basinhopping());

в) реализацию алгоритмов минимизации методами наименьших квадратов(leastsq()) и приближения с помощью кривых (curve_fit());

г) реализацию методов определения минимумов (minimize_scalar()) и корней (newton()) скалярных функций одной переменной.

Рассмотрим применение различных методов оптимизации к более интересному случаю, а именно, нахождению оптимума функции Розенброка. Функция Розенброка – невыпуклая функция, предложенная Ховардом Розенброком в 1960 г. для оценки производительности алгоритмов оптимизации.

Данная функция от двух переменных задаётся следующим уравнением:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \ .$$

Её график представлен на рисунке 20.

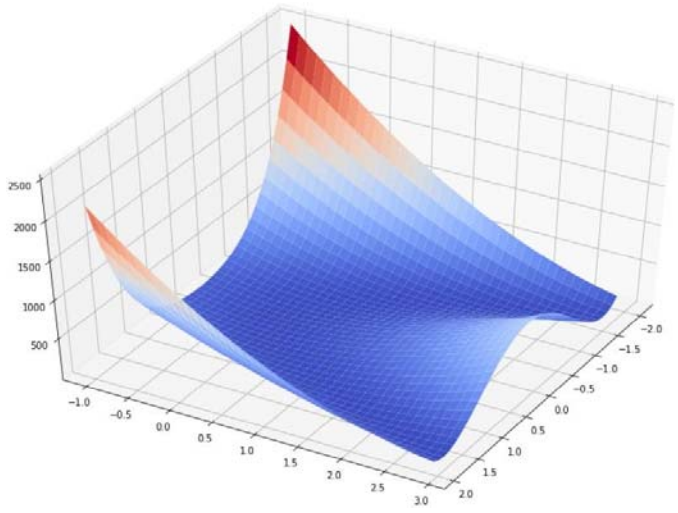


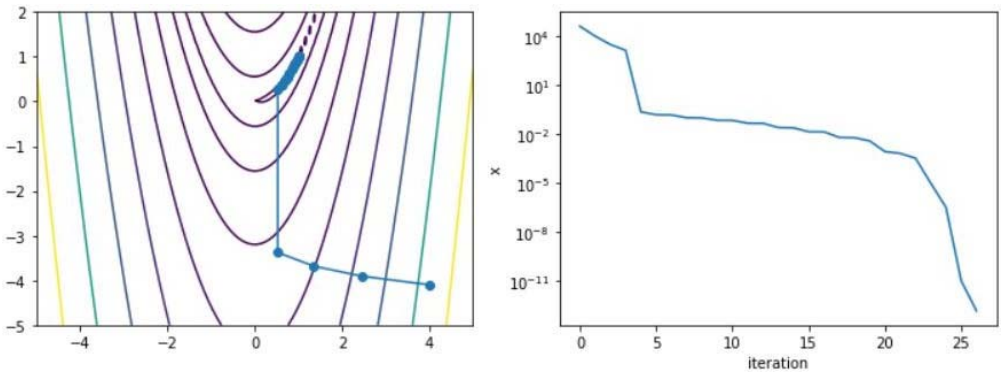
Рисунок 6.3 – График функции Розенброка от двух переменных

Функция Розенброка имеет минимум 0 в точке (1; 1).

Метод сопряженных градиентов

– итерационный метод для безусловной оптимизации в многомерном

пространстве, являющийся расширением идей метода Ньютона, который ищёт экстремум квадратичной формы, полученной из целевой функции. Проиллюстрируем результат применения этого метода для нахождения минимума функции Розенброка. Будем искать экстремум функции Розенброка изначальной точки (4; -4,1) функцией scipy.optimize.minimize() с аргументом method=’Newton-CG’ (рисунок 6.4).



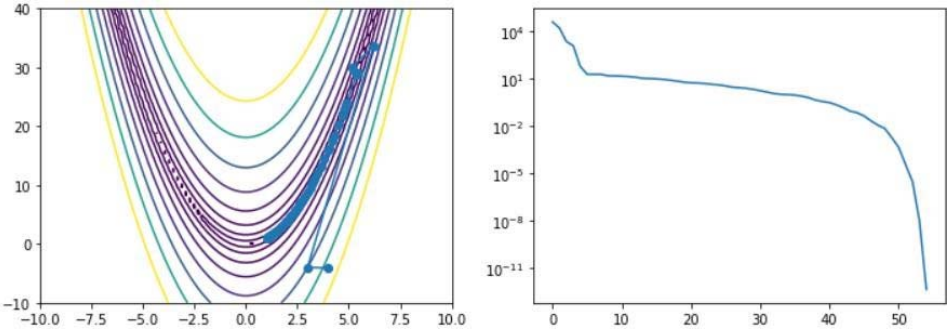
а б

Рисунок 6.5 – Поиск минимума функции Розенброка методом сопряженных градиентов: последовательность приближенных решений на различных итерациях алгоритма на 2D карте функции (а); график зависимости значения функции от номера итерации (б)

На 26-й итерации алгоритма было получено решение (0,99999963; 0,99999926).

Алгоритм Бройдена-Флетчера-Гольдфарба-Шанно (BFGS)

Алгоритм BFGS - один из наиболее популярных квазиньютоновских методов. В квазиньютоновских методах не вычисляется напрямую гессиан



(матрица вторых производных) функции, требуемый для расчёта шага оптимизационного алгоритма в многомерном пространстве. Вместо этого гессиан оценивается приближенно, исходя из сделанных до этого итераций. Посмотрим на решение той же задачи оптимизации, что и при применении метода сопряженных градиентов. Воспользуемся функцией scipy.optimize.minimize() с аргументом method=’BFGS’ (рисунок 6.6).

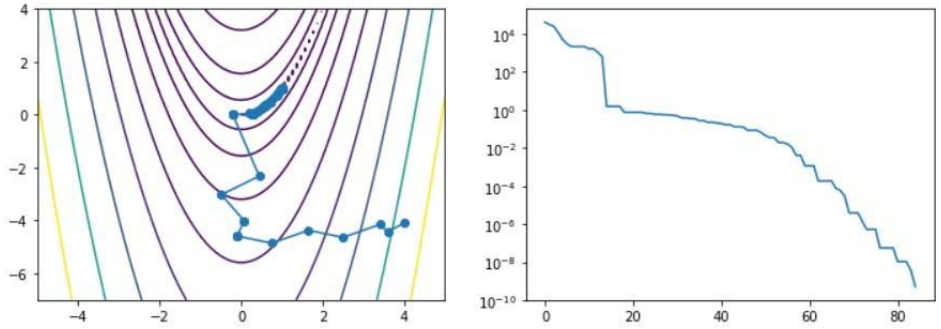
а б

Рисунок 6.5 – Поиск минимума функции Розенброка алгоритмом BFGS: последовательность приближенных решений на различных итерациях алгоритма на 2D карте функции (а); график зависимости значения функции ошибки от номера итерации (б)

На 54-й итерации алгоритма было получено решение (0,99999939; 0,99999876). Из рисунка 6.5 (а) видно, что на втором шаге алгоритм «шагнул» в неправильном направлении, но затем последовательно начал движение к искомому минимуму функции.

Симплекс-метод Нелдера-Мида

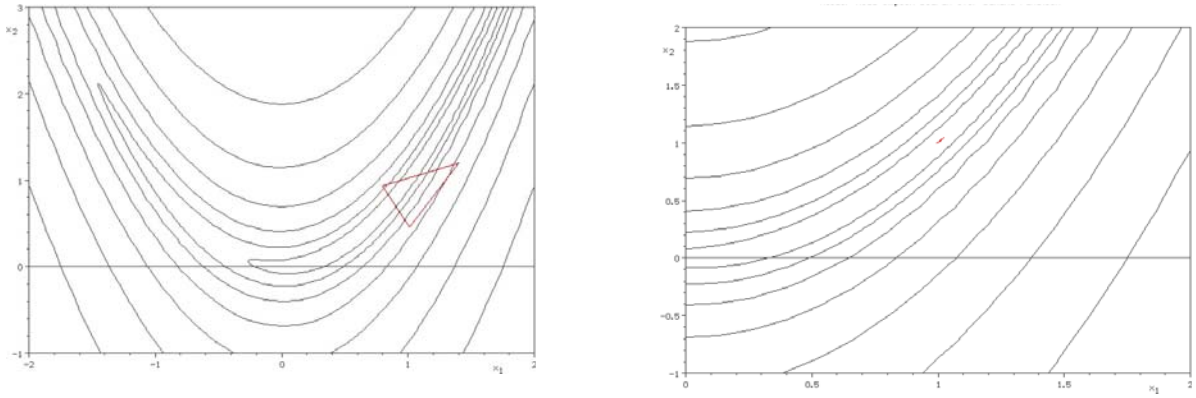
Симплекс метод, называемый также методом деформируемого многогранника, является методом безусловной оптимизации функции нескольких переменных, не использующий градиентов функции. Это свойство позволяет применять его к негладким функциям. Суть метода заключается в последовательном перемещении и деформировании симплекса (чаще всего, треугольника) вокруг точки экстремума. Проиллюстрируем результат метода на той же задаче оптимизации. Будем использовать функцию scipy.optimize.minimize с аргументом method=’nelder-mead’. На рисунке 6.5 визуализирован поиск точки минимума, а на рисунке 6.6 – примеры симплексов в начале и конце



работы итерационного алгоритма: посредством последовательного уточнения положения экстремума симплекс «шагает» по гиперплоскости, одновременно уменьшаясь в размерах, всё более приближаясь к целевой точке.

а б

Рисунок 6.6 – Поиск минимума функции Розенброка симплекс-методом Нелдера-Мида: последовательность приближенных решений на различных итерациях алгоритма на 2D карте функции (а); график зависимости значенияфункции ошибки от номера итерации (б)



а б

Рисунок 6.7 – Примеры симплексов (красные линии) на 2D карте функции Розенброка (черные линии) в начале (а) и конце (б) работы симплекс-методаНелдера-Мида

На 85-й итерации алгоритма было получено решение (0,99998846; 0,99997494).

Градиентный спуск

Градиентный спуск – алгоритм, используемый для нахождения (в общем случае) локальных минимумов дифференцируемой функции. Для заданной дифференцируемой функции $f(x)$ алгоритм позволяет найти x^* такой, что $f'(x^*) = 0$ и x^* является точкой минимума $f(x)$. Функция может

обладать несколькими локальными минимумами $x_1^*, x_2^*, \dots, x_k^*$, и алгоритм градиентного спуска будет сходиться к одному из них, в зависимости от выбора начальной точки и скорости обучения.

Предположим, что $f(x)$ – дифференцируемая функция от одной переменной. При текущем значении x_1 градиентный спуск описывает направление для сходимости к локальному минимуму x^*, где $f'(x^*) = 0$. Следующая точка определяется как $x_2 = x_1 - \lambda f'(x_1)$, где λ – коэффициент скорости обучения. Можем записать данную формулу в обобщенном виде:

$$x_{t+1} = x_t - \lambda f'(x_t),$$

t – номер итерации. При заданном x_1 и большом количестве итераций T алгоритм генерирует последовательность точек $x_1, x_2, \dots, x_T$, где $x_T \approx x^*$. Таким образом, x_{t+1} сходится к x^* при очень большом t. Стоит заметить, что при сходимости алгоритма $f'(x_t) \approx 0$, и, следовательно, $|x_{t+1} - x_t| \approx 0$. Таким образом, общепринятым критерием остановки работы градиентного спуска является уменьшение величины $|x_{t+1} - x_t|$ до какого-либо малого значения.

Например, можно задать алгоритму работать до момента, когда $|x_{t+1} - x_t| < 0$.

Теперь пусть имеется функция, дифференцируемая от нескольких переменных $f(x_1, x_2, \dots, x_n)$. Наша цель – с помощью градиентного спуска найти точку минимума $(x_1^*, x_2^*, \dots, x_n^*)$. Функция имеет частные производные, хранящиеся в векторе градиента $\left(\frac{df(x)}{dx_1}, \frac{df(x)}{dx_2}, \dots, \frac{df(x)}{dx_n}\right)$. Направление градиента задает направление наибольшего подъема, а длина вектора – угол наклона. Можно легко выписать обобщенное выражение градиентного спуска для функций от нескольких переменных:

$$\begin{bmatrix} x_1^{t+1} \\ \vdots \\ x_n^{t+1} \end{bmatrix} = \begin{bmatrix} x_1^t \\ \vdots \\ x_n^t \end{bmatrix} - \lambda \begin{bmatrix} \frac{df(x_1^t)}{dx_1} \\ \vdots \\ \frac{df(x_n^t)}{dx_n} \end{bmatrix},$$

где t – номер итерации. В данном случае критерий остановки связан с евклидовым расстоянием между текущей и предыдущей итерациями, например,

можно задать: $\sqrt{(x_1^{t+1} - x_1^t)^2 + \cdots + (x_n^{t+1} - x_n^t)^2} < 0{,}001$.

Пример: минимизация функции одной переменной

Целевая функция: $f(x) = 0{,}1x^2 + \text{in}(0{,}1x^2)$ (рисунок 6.8).

Алгоритм градиентного спуска требует расчета первой производной функции: $f'(x) = 0{,}2x + 0{,}2x\cos(0{,}1x^2)$.

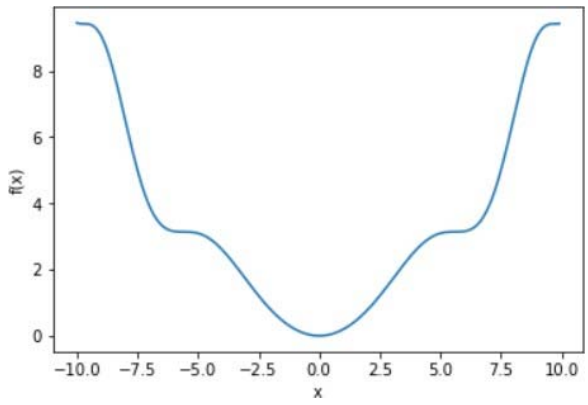


Рисунок 6.8 – График функции $f(x) = 0{,}1x^2 + \sin(0{,}1x^2)$

При коэффициенте скорости обучения $\lambda = 1$ и при начальной точке $x_0 = 8$ получаем схождение к минимуму функции, представленное на рисунке 6.9.

Порог для критерия остановки был задан $\varepsilon = 0{,}001$, в результате после 20-й итерации было найдено решение $x \approx 0{,}0011$.

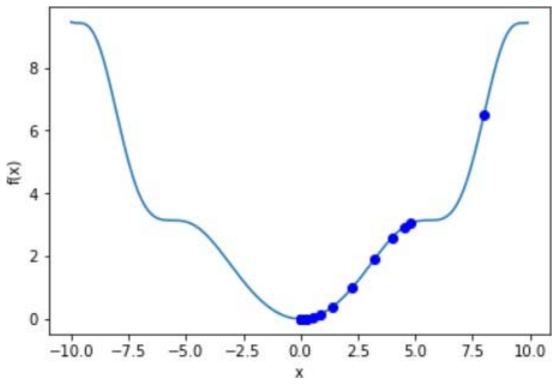


Рисунок 6.9 – Приближенные решения на различных итерациях градиентногоспуска при поиске минимума функции $f(x) = 0{,}1x^2 + \sin(0{,}1x^2)$

Пример: минимизация функции нескольких переменных

Целевая функция: $f(x,y) = x^2 + y^2 + 1$ (рисунок 6.10), – частные производные которой $\frac{df(x)}{dx} = f_x = 2x$ и $\frac{df(y)}{dy} = f_y = 2y.$ $= 2y.$

$f(x,y) \geq 1$ и $f(0,0) = 1$, то есть $(0;0)$ – глобальный минимум функции.

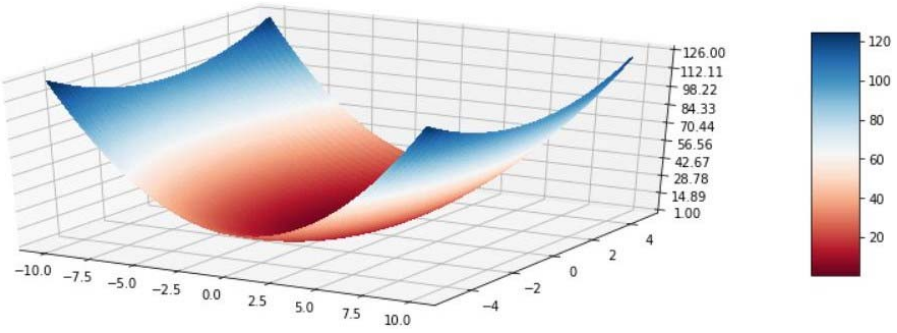


Рисунок 6.10 – График функции $f(x,y) = x^2 + y^2 + 1$

Для работы алгоритма были заданы параметры: $\lambda = 0{,}2,\ x_0 = (6;2)$, $\varepsilon = 0{,}001$.

Схождение градиентного спуска к точке минимума на 2D карте функции визуализировано на рисунке 6.11.

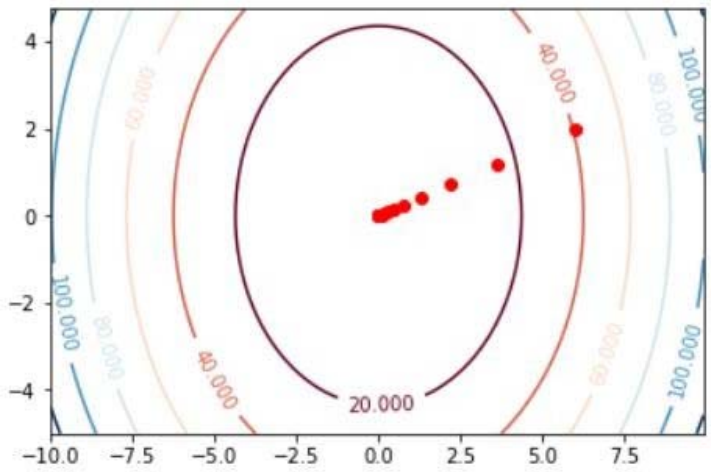


Рисунок 6.11 – Приближенные решения на различных итерациях градиентногоспуска при поиске минимума функции $f(x) = x^2 + y^2 + 1$

Стохастический градиентный спуск

Применение алгоритма градиентного спуска на тренировочном датасете большого размера может вызывать затруднения, связанные со скоростью сходимости алгоритма и требуемых вычислительных мощностей. На одномшаге классического градиентного спуска выполняется обновление параметров модели в соответствии с правилом:

$$\theta_t \coloneqq \theta_{t-1} - \eta \nabla_\theta J(\theta_{t-1}),$$

где η – коэффициент скорости обучения, θ – параметры модели.

Необходимо пройтись по всем данным, чтобы рассчитать градиент $\nabla_\theta J(\theta_{t-1})$ и новое значение θ_t. В случае большого объема данных это может быть вычислительно дорогой операцией. Кроме того, такой подход не может обойти проблему «падения» в локальные минимумы. Стохастическаямодификация алгоритма позволяет отчасти решить обе эти задачи.

Стохастический градиентный спуск производит обновление параметров на каждом поднаборе данных (i):

$$\theta_t \coloneqq \theta_{t-1} - \eta \nabla_\theta J\left(\theta_{t-1}^{(i)}\right)$$

,

где $J(\theta)$ – целевая функция (функция ошибок), θ – параметры модели, η – коэффициент скорости обучения. Для этого, как видно из формулы, для обновления весов не требуется считать градиент по всему набору данных, что позволяет значительно экономить вычислительные ресурсы. При этом, если этот поднабор данных содержит только один элемент, то мы получаем классический вариант стохастического градиентного спуска, а если 1 < i < N, то это промежуточный вариант, который в английской литературе называется “mini batch gradient descent”. Размер «минибатча» - поднабора данных – в каждом случае нужно подбирать отдельно в зависимости от свойств оптимизируемой функции и других требований к процессу обучения модели.

Градиентный спуск с инерцией (Momentum)

Данный метод ускоряет движение градиентного спуска в сторонуэкстремума и уменьшает осцилляции в неверных направлениях. В алгоритме используется доля β вектора градиента от предыдущей итерации и доля $(1 - \beta)$текущего вектора градиента. Это полный аналог так называемой функции экспоненциального сглаживания:

$$V_t = \beta V_{t-1} + (1 - \beta) g_t$$
$$\theta_t \coloneqq \theta_{t-1} - \eta V_t,$$

где g_t – градиент на текущей итерации. Обычно β устанавливают равным $0{,}9$, что позволяет в большей степени учитывать инерцию движения, накопленного за предыдущие итерации, и в меньшей степени обращать внимание на новые изменения. Отсюда и аналогия с физикой: мы «по инерции» движемся в выбранном направлении, постепенно уточняя его, получая информацию от новых данных.

RMSprop

Алгоритм RMSprop (Root Mean Square propagation) обновляет параметрыс учётом квадратов градиентов на текущем шаге и имеет несколько изменённое правило обновления весов:

$$V_t = \beta V_{t-1} + (1-\beta) g_t^2$$

$$\theta_t := \theta_{t-1} - \eta \frac{g_t}{\sqrt{V_t} + \varepsilon}.$$

где s – маленькая добавка во избежание деления на 0. Адаптивный коэффициент скорости обучения с квадратным корнем в знаменателе в последней формуле обеспечивает умеренный сдвиг в нужном направлении к точке локального минимума. Другими словами, чем больше «накопленный» градиент V_t «текущего» градиента g_t, тем меньше будут изменяться веса на каждом шаге.

Adam

Adam (Adaptive momentum algorithm) – один из самых популярных оптимизационных алгоритмов и содержит в себе элементы методов Momentum и RMSprop. В нём производится расчет адаптивного коэффициента скорости обучения и сохраняются экспоненциальные скользящие средние градиентов и квадратов градиентов, а также могут вводиться скорректированные градиенты для «замедления» скорости движения к экстремуму по мере увеличения номера итерации t:

$$V_t^{corr} = \frac{V_t}{1-\gamma^t}$$

,

где γ – число от 0 до 1. Гиперпараметры алгоритма Adam: скорость обучения η, коэффициенты β_1, β_2, число ε.

Скользящие средние градиентов и квадратов градиентов, V_t и S_t соответственно, рассчитываются следующим образом:

$$V_t = \beta_1 V_{t-1} + (1 - \beta_1) g_t$$

$$S_t = \beta_2 S_{t-1} + (1 - \beta_2) g_t^2$$

Обновление весов модели в алгоритме Adam производится по формуле:

$$\theta_{t+1} := \theta_t - \frac{\eta}{\sqrt{S_t^{corr}} + \varepsilon} V_t$$

Предложенные авторами алгоритма значения (которые в большинстве фреймворков глубокого обучения можно менять вручную): $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $s = 10^{-8}$.

Пример: распознавание цифр

Рассмотрим пример применения рассмотренных градиентных алгоритмов оптимизации для настройки весов полносвязной нейронной сети. Построим модель распознавания цифр по графическим черно-белым изображениям размера 28 на 28 пикселей (рисунок 29), взятым из базы данных MNIST.

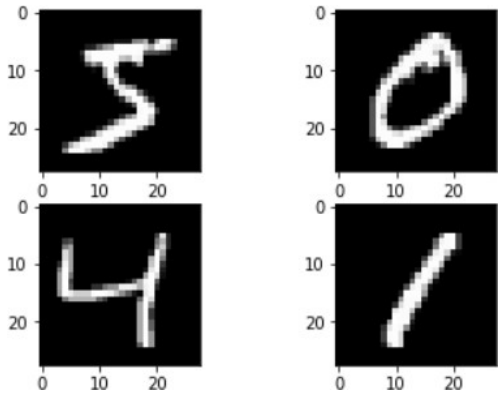
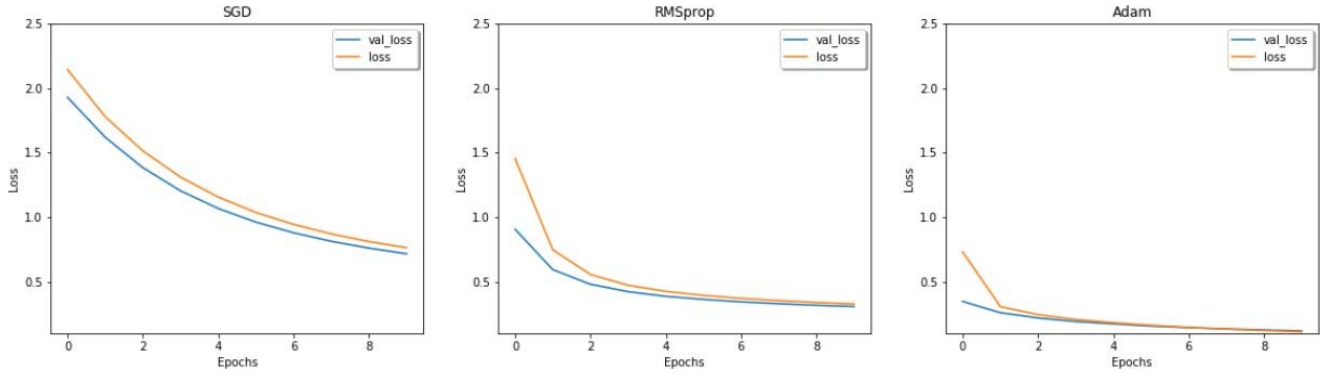


Рисунок 6.12 – Примеры изображений из базы данных MNIST

Для обучения будем использовать простую нейронную сеть с одним скрытым слоем. Проведём обучение три раза, с разными алгоритмами оптимизации из tensorflow.keras.optimizers: SGD() (стохастический градиентный спуск), RMSprop() и Adam(). Значение коэффициента скорости обучения 5 возьмём равным 0,0001. На рисунке 6.13 можно видеть динамику изменения функции ошибок за 10 эпох обучения модели с разными оптимизаторами.

Можно видеть преимущество использования алгоритма Adam для данной задачи.



а б в

Рисунок 6.13 – Значения функции ошибок по эпохам обучения модели распознавания цифр при использовании в качестве алгоритма оптимизации:стохастического градиентного спуска (а), RMSprop (б), Adam (в)

Задания к лабораторной работе

Задание 1. Выберите любой датасет из открытых источников

Задание 2. Реализуйте функцию для алгоритма обратного распространенияошибки в виде градиентного спуска для трёхслойной полносвязной сети с 32-16-10 нейронами в слоях соответственно и среднеквадратичной функции ошибок для задачи бинарной классификации.

Задание 3. Визуализируйте процесс обучения сети, построив график изменения лосс-функции.

Задание 4. Реализуйте алгоритм стохастического градиентного спуска и егомодификации:

а) метод моментов;

б) RMSprop;

в) Адам.

Задание 5. Сделайте сравнительный анализ точности решения оптимизационной задачи с помощью методов, реализованных в задании 4.

Контрольные вопросы

1. Почему стохастический градиентный спуск работает быстрее, чем

классический градиентный спуск?

2. Какие существуют методы обхода локальных минимумов?

3. Чем, по вашему мнению, должен определяться размер mini-batch для реализации градиентного спуска в конкретном случае?

4. Обучите свёрточную нейронную сеть разными алгоритмами оптимизации для решения задачи из текста лабораторной работы. Какой алгоритм оптимизации для настройки весов свёрточной сети сработает лучше в этом случае и почему?

Литература

1) Николенко С., Кадурин А., Архангельская Е. Глубокое обучение.Погружение в мир нейронных сетей. – Изд-во «Питер». – 2018. – 476 с.;

2) Shiliang Sun, Zehui Cao, Han Zhu, and Jing ZhaoA, Survey of OptimizationMethods from a Machine Learning Perspective, 2019.

3) https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html

Лабораторная работа 7

Свёрточные сети и работа с изображениями

Цель: получение навыков реализации свёрточных нейронных сетей и метода переноса обучения.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: реализация свёрточных нейронных сетей и метода переноса обучения.

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

Архитектура свёрточной нейронной сети

При попытке применить обычную полносвязную нейронную сеть для обработки изображений с целью решения какой-нибудь задачи, например, классификации изображений собак и кошек, мы столкнёмся со следующими проблемами:

Потеря важной информации при преобразовании изображения в вектор Действительно, при подаче на вход полносвязной сети изображения в виде одномерного вектора (например, с применением метода reshape() библиотеки numpy) мы неизбежно потеряем информацию о взаимном расположении объектов на изображении и их топологической структуре;

Вычислительная неэффективность: действительно, при преобразовании одноканального изображения размером всего лишь 28 x 28 пикселей размерность входного вектора будет (784,1), что с точки зрения полносвязной сети будет соответствовать 784 признакам изображения. С учётом того, что для картинок размером больше 1 Мб размерность входного вектора превысит 10^6, становится очевидной неэффективность использования такого подхода.

В свёрточной нейронной сети основным составным блоком является слойсвёртки, состоящий из нескольких (обычно от единиц до нескольких десятков) фильтров, каждый из которых настраивается в процессе обучения для

выявления характеристических признаков на изображении. Эти фильтры в режиме «скользящего окна» проходят по всему изображению, записывая результат свёртки с элементами изображения в соответствующую ячейку выходного изображения. Выходное изображение в данном случае называется «картой признаков», так как соответствует выделенным с помощью данного фильтра признакам. Сама операция свёртки, которая реализуется с каждымфильтром в свёрточном слое, выражается в попарном перемножении элементовфильтра с элементами входного изображения, причем размер окна в точности равен размеру фильтра.

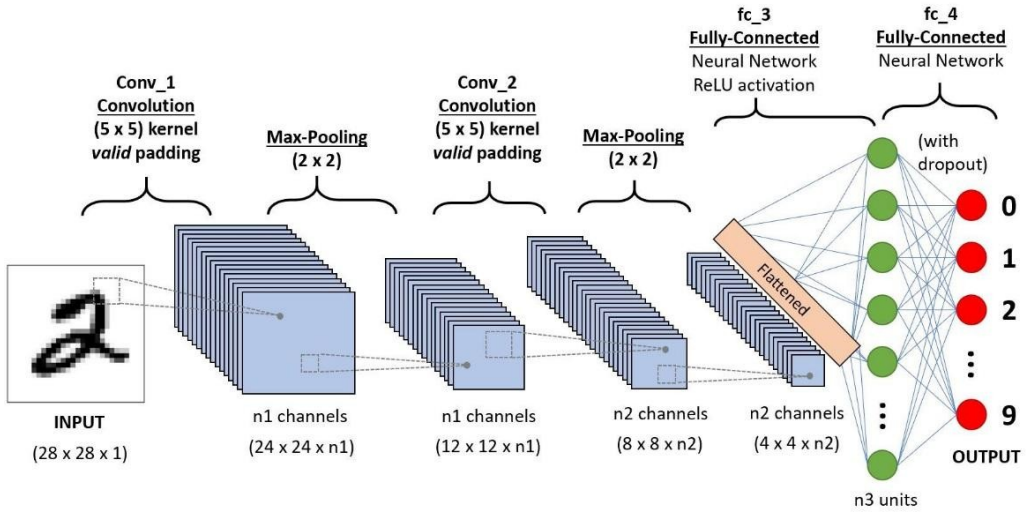


Рисунок 7.1 – Пример классической архитектуры сверточной нейронной сети

Результат операции свёртки можно выразить следующей формулой:

$$(f * g)[m.n] = \sum_{k,l} f[m-k, n-k] * g[k,l]$$

где f – исходная матрица изображения; g – фильтр свёртки.

Фильтры в первых слоях сети будут активироваться базовыми признаками, наподобие линий, углов, блоков, при этом при движении к более глубоким слоям свёрточной сети фильтры в них будут выделять всё более сложные признаки (см. рисунок 7.4). Размер ядра обычно берут в пределах от 3x3 до

7x7. Если размер ядра маленький, то оно не сможет выделить какие-либопризнаки, если слишком большое, то увеличивается количество связей между нейронами. Каждый фильтр представляет собой матрицу весов, настраиваемых в процессе обучения модели: это одна из главных особенностей сверточной нейронной сети, заключающаяся в принципе «shared weights» (общих весов), которая позволяет сократить число связей и позволяет находить один и тот же признак по всей области изображения.

При этом в зависимости от метода обработки краев исходной матрицы результат свёртки может быть меньше исходного изображения («valid»), такого же размера («same»). На практике, чтобы не терять информацию с пограничныхпикселей изображения, чаще используют свёртку с сохранением размера входного изображения, для этого исходную матрицу дополняют одним или несколькими слоями пикселей. Эта операция называется паддинг (“padding”) и описывается параметром p, который отвечает за «толщину» добавленного слоя (см. рисунок 32). Чаще всего дополнительные пиксели инициализируются нулевыми значениями.

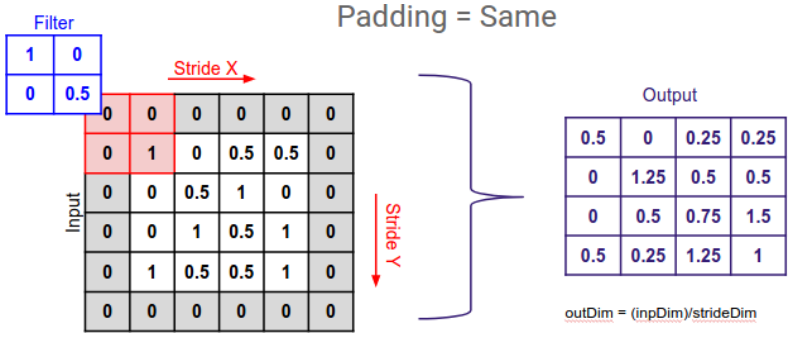


Рисунок 7.2 – Иллюстрация операции «padding»

Размер карты признаков можно вычислить по следующей формуле:

$$n_{out} = \left\lceil \frac{n_{in} + 2p - k}{s} \right\rceil + 1,$$

где n_{in} – размер входного изображения; n_{out} – размер выходного изображения (карты признаков); k – размер фильтра; p – размер паддинга; s – страйд.

За слоем свёртки после применения функции активации к получившимся картам признаков (в свёрточных сетях – это чаще всего ReLU и её модификации) почти всегда следует слой субдискретизации или пуллинга (см. рисунок 33). Задача этого блока - уменьшение размерности карт признаков предыдущего слоя. Это делается для снижения числа параметров модели и во избежание переобучения. Также при уменьшении размерности карт признаков мы осуществляем переход к другому масштабу признаков, что в конечном итоге позволяет переходить от точек, линий и пятен на первых слоях к высокоуровневым признакам, содержащим части реальных объектов на последних слоях сети. Размер окна пуллинга – обычно 2x2, приэтом применяют два варианта выбора значения в окне: выбор максимального элемента - “MaxPooling”) либо среднего элемента - (“AveragePooling”), котороезатем записывают в соответствующую ячейку выходной матрицы.

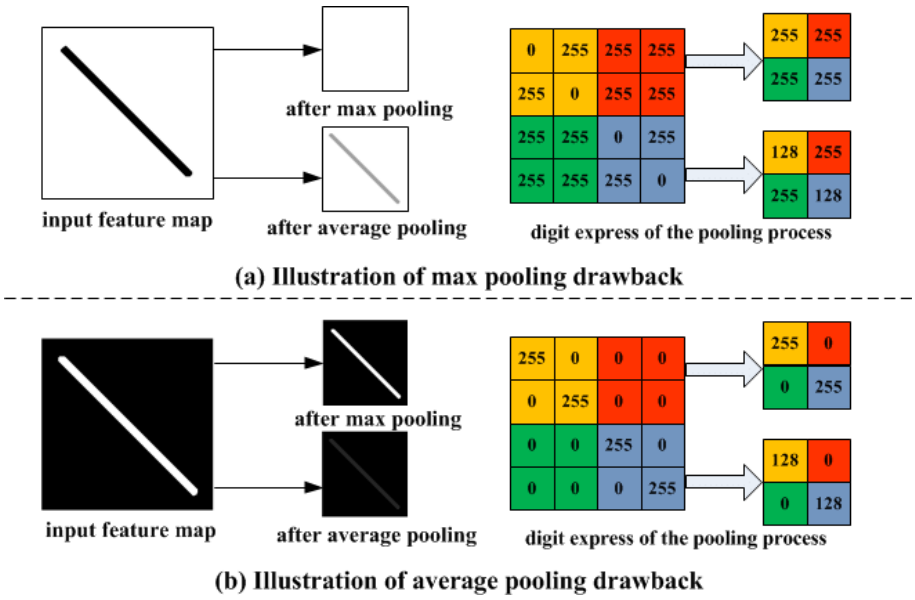


Рисунок 7.3 – Иллюстрация операций субдискретизации MaxPooling и AveragePooling

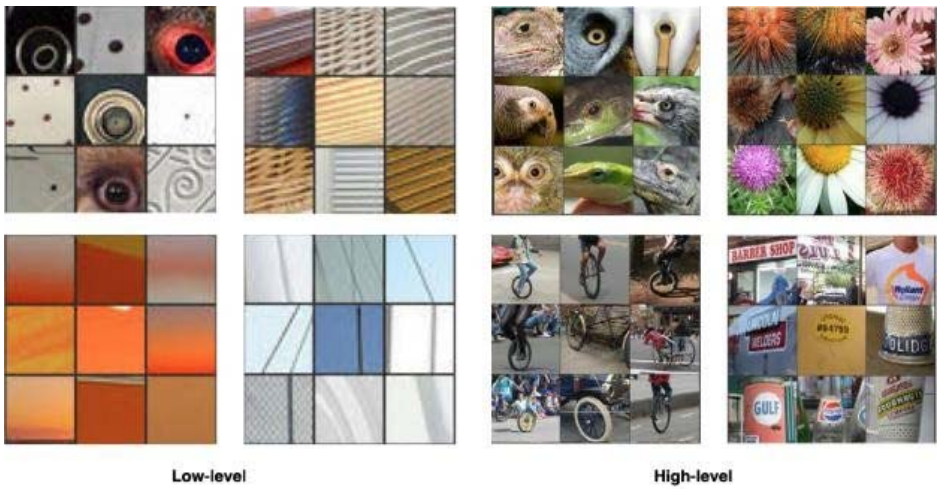


Рисунок 7.4 – Признаки низкого уровня (а), максимизирующие активацию нейронов на первых слоях свёрточной сети, и признаки высокого уровня (б) (максимизирующие активацию нейронов на последних слоях свёрточной сети)

За последние несколько лет в задачах классификации изображений сверточные нейросети добились точности, сравнимой с точностью, достигаемой человеческим мозгом, а в некоторых задачах они существенно превосходят человеческий мозг. Если первая большая нейросеть из группы университета Торонто, показавшая результат, превзошедший лучшие модели классического компьютерного зрения, содержала чуть более десятка слоев, то современные свёрточные архитектуры содержат свыше сотни слоёв. Разумеется, для обучения таких глубоких сетей, показывающих сейчас state-of- the-art результаты в задачах анализа изображений, требуется огромное вычислительное время с использованием десятков GPU-процессоров.

Перенос обучения

В реальных задачах компьютерного зрения часто встречаются ситуации, когда требуется, например, обучить классификатор изображений с объектами, которые не присутствуют в базовых датасетах, использовавшихся для обучения state-of-the-art моделей. В таких случаях можно использовать уже настроенные веса моделей, обученных на большом наборе данных, таких как

ImageNet, и затем осуществлять тонкую настройку дополнительных параметров на новые данные, относящиеся к текущей задаче. Чем больше новые данные будут отличаться от тех, на которых обучалась базовая модель, тем больше параметров (слоев нейронной сети) необходимо будет «переобучить», чтобы получить хорошую точность в новом домене данных. Интуиция здесь заключается в том, что модель уже научилась выделять необходимые признаки в большом наборе данных, ее нужно только «поднастроить» для выполнения конкретной задачи запоминания новых высокоуровневых признаков целевого домена. Такой подход носит название «перенос обучения» (“transfer learning”) ишироко применятся в современном машинном обучении. Метод позволяет эффективно использовать веса «больших» моделей, для переобучения которых понадобилось бы использование существенных вычислительных ресурсов. Большинство фреймворков глубокого обучения позволяет загружать веса предобученных моделей для переиспользования и дообучения на целевом домене данных. На рисунке 35 проиллюстрирована схема метода переноса обучения.

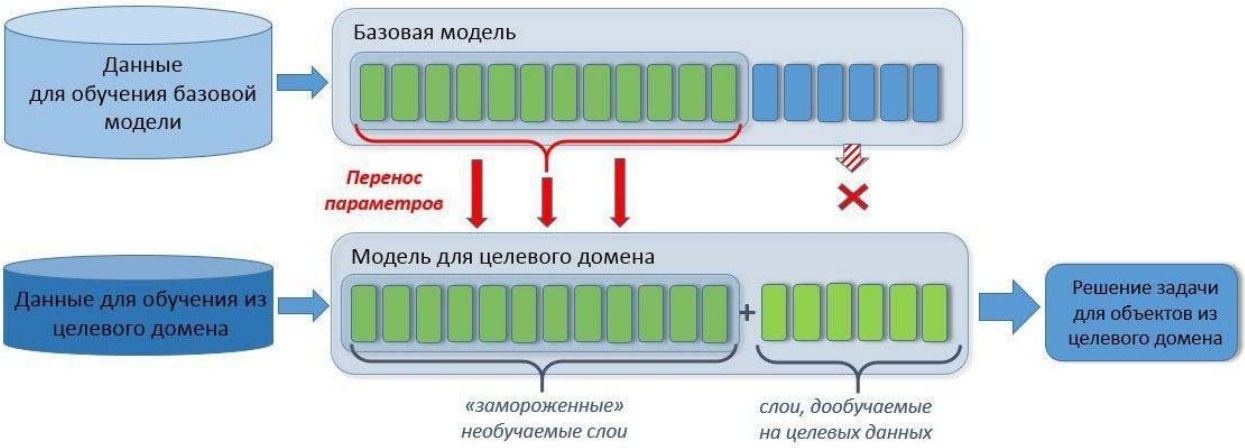


Рисунок 7.5 – Схема переноса обучения

Задания для лабораторной работы

Задание 1. Скачайте данные для обучения моделей, используя базу открытых датасетов домашних животных.

Задание 2. Постройте модель классификации собранных в датасете

изображений на собак и кошек. Для этого последовательно реализуйте:

а) полносвязую сеть с тремя скрытыми слоями для классификации изображений (на вход – одномерный вектор)

б) сверточную нейронную сеть с двумя блоками свёртки и субдискретизации для той же цели.

Задание 3. Реализуйте перенос обучения для моделей VGG19 и ResNet, воспользовавшись весами предобученных моделей, «заморозив» полносвязные слои и переобучив их на новых данных.

Задание 4. Сравните производительность моделей.

Задание 5. Увеличьте число эпох обучения для моделей (а) и (б) и постройте кривые обучения, демонстрирующие явление переобучения.

Контрольные вопросы

1. Какой будет размер RGB-изображения 128x128x3 на выходе блока свёрточной сети с размером фильтра (kernel) (3x3) и слоя субдискретизации (MaxPooling) с окном (2x2) с шагом (stride) равным 2?
2. Сколько обучаемых параметров у сверточной нейронной сети,изображенной на рисунке 33?
3. Зачем в свёрточной сети используются слои субдискретизации (пулинга)? Как бы работали модели без его использования?

4. Почему свёрточным сетям нужны тысячи изображений для достижения приемлемого качества классификации, тогда как человеку достаточно всего нескольких примеров?

Литература

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org]

2. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. Погружение в мир нейронных сетей. – Изд-во «Питер». – 2018. – 476 с.

3. Портал – [https://towardsdatascience.com/a-comprehensive-guide-to- convolutional-neural-networks-the-eli5-way-3bd2b1164a53]

Лабораторная работа № 8

Анализ и предсказание временных рядов

Цель работы: является получение навыков анализа временных рядов, оценки стационарности ряда и классических методов прогнозирования временных рядов.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: анализ временных рядов, оценка стационарности рядов, методы прогнозирования временных рядов.

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

По определению временной ряд – последовательные значения какого-либо признака $\{y_1, \ldots, y_T\}$, $y_t \in R$, измеренные через постоянные временные интервалы. Временные ряды встречаются во всех сферах человеческой деятельности – от экономики до астрофизики, поэтому владение навыками анализа и, в особенности, прогнозирования временных рядов принципиально важно для эффективной работы аналитика данных. Среди главных характеристик временных рядов выделяют: тренд - плавное долгосрочное изменение уровня ряда; сезонность - циклические изменения уровня ряда с постоянным периодом; цикл - изменения уровня ряда с переменным периодом (цикл жизни товара, экономические волны, периоды солнечной активности); шум (или ошибка) - непрогнозируемая случайная компонента ряда. Примеры различных временных рядов представлены на рисунке 8.1.

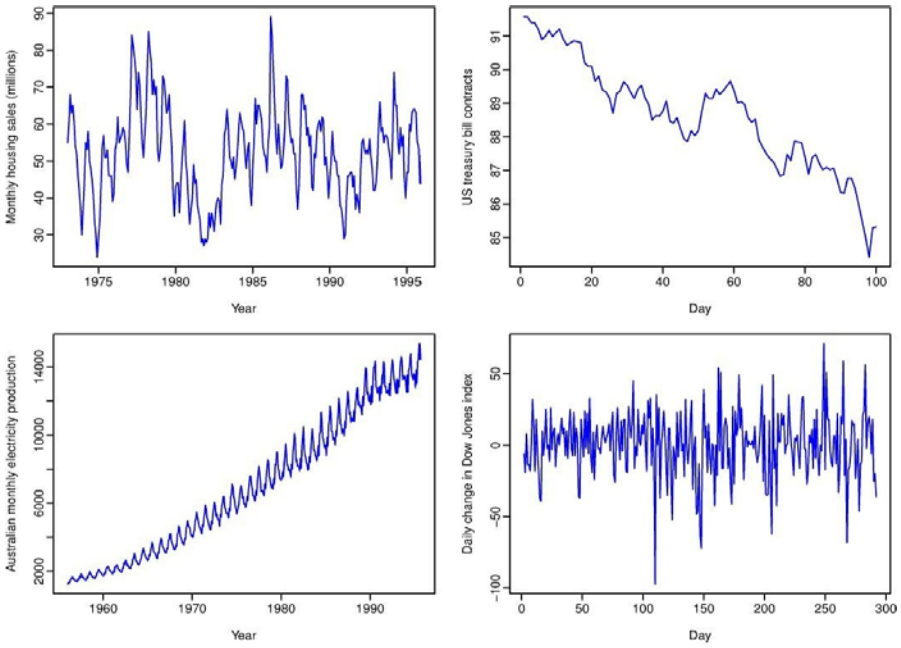


Рисунок 8.1 – Примеры временных рядов [3]

Важным свойством ряда также является его стационарность. Ряд $(y_1, \ldots, y_T)$ стационарен, если для люб s распределение $(y_1, \ldots, y_T)$ не зависит от t, т. е. его свойства не зависят от времени. Следует отметить, что ряды с трендом или сезонностью нестационарны, а ряды с непериодическими циклами стационарны, поскольку нельзя предсказать заранее, где будут находиться максимумы и минимумы их значений.

Для проверки временного ряда на стационарность часто используется статистический тест Дики-Фуллера, заключающийся в проверке нулевойгипотезы о нестационарности временного ряда путём поиска единичного корня в соответствующем авторегрессионном уравнении первого порядка. Для приведения временного ряда к стационарности применяют различные техники, в том числе дифференцирование временного ряда (преобразование ряда к ряду последовательных разностей), сезонное дифференцирование (преобразование ряда к ряду разностей между значениями, отстоящими друг от друга на величину периода) и преобразование Бокса–Кокса, которое выражается следующей формулой:

$$x_i(\lambda) = \begin{cases} \dfrac{x_i^{\lambda} - 1}{\lambda}, & \lambda \neq 0 \\ \ln(x_i), & \lambda = 0 \end{cases}$$

где λ – параметр, подбираемый исходя из максимизации правдоподобия.

При $\lambda = 0$ преобразование Бокса-Кокса представляет собой операцию логарифмирования исходного ряда.

Модели *ARIMA* (или в общем виде, с учётом сезонной («seasonal»)составляющей, *SARIMA*) – Autoregressive Integrated Moving Average – классодних из самых популярных классических моделей прогнозирования временных рядов. Такие модели (интегрируемые модели авторегрессии и модели скользящего среднего) – достаточно гибкие и могут описыватьмножество характеристик ряда. В модели авторегрессии каждое значение ряда находится в линейной зависимости от *p* предыдущих значений. Модель скользящего среднего же предполагает, что в ошибках модели за *q* предшествующих шагов сосредоточена информация о предыстории ряда.

В зависимости от свойств изучаемого показателя, модели *ARIMA* могут включать в себя сразу обе модели, или каждую по отдельности (*AR* и *MA*). Если добавить в ряд модели *ARMA P* слагаемых авторегрессии, отстоящих друг от друга на интервал, равный значению периода ряда, и, аналогично, *Q* слагаемыхскользящего среднего, то модель будет включать в себя сезонные компоненты и называться *SARMA(p.P,q,Q)*.

Если процесс оказывается нестационарным и для приведения его к стационарному виду потребовалось взять несколько разностей, то модель становится моделью *SARIMA(p,d,q),* где *d* – порядок разности. Если бралось несколько сезонных производных, то в модель добавляется параметр *D*, отвечающий за число взятых сезонных производных. В общем виде модель *SARIMA(p,P.q,Q,d,D)* для предсказания на один шаг вперёд выглядит следующим образом:

$$y_t = \alpha + \theta_1 y_{t-1} + \theta_2 y_{t-2} + \cdots + \theta_p y_{t-p} + \epsilon_t + \varphi_1 \epsilon_{t-1} + \varphi_2 \varepsilon_{t-2} + \cdots$$

$$+ \varphi_q \epsilon_{t-q} + \theta_S y_{t-S} + \theta_{2S} y_{t-2S} + \cdots + \theta_{PS} y_{t-PS} + \varphi_S \epsilon_{t-S} + \varphi_{2S} \varepsilon_{t-2S} + \cdots + \varphi_{QS} \epsilon_{t-QS}.$$

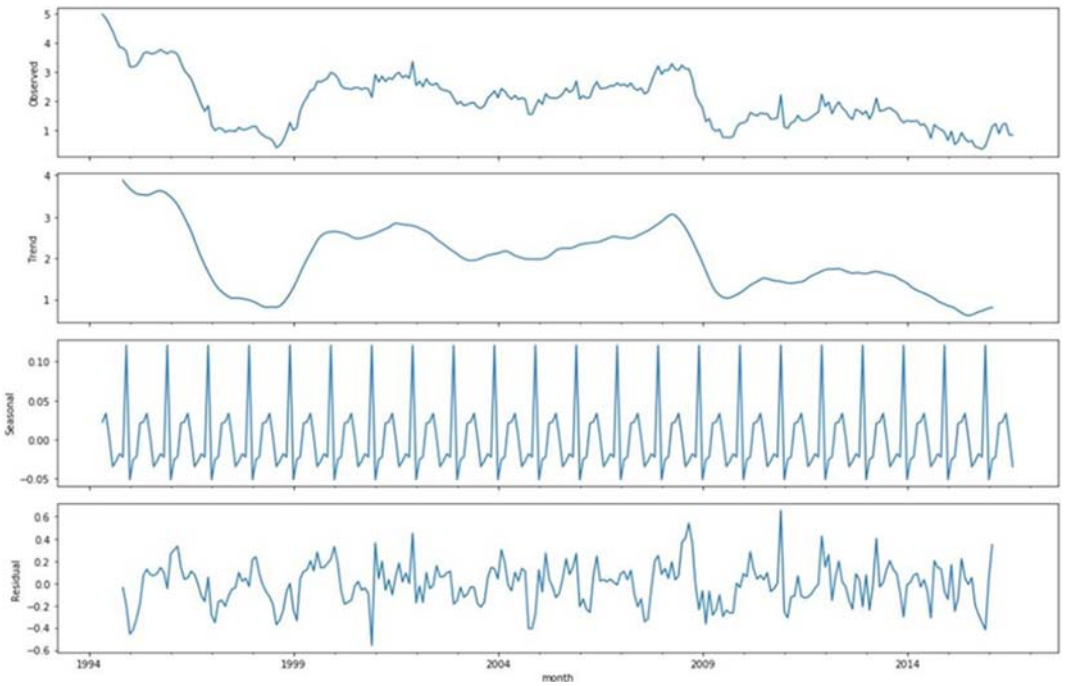


Рисунок 8.2 – Декомпозиция временного ряда. Сверху вниз: исходный ряд,тренд, сезонная компонента, остатки (шум)

Параметры $\{\theta_i\}$, $\{\varphi_i\}$ и α настраиваются в процессе обучения модели.

В общем случае для сравнения различных моделей прогнозирования с точки зрения баланса между точностью предсказания и сложностью (количеством параметров модели) применятся критерий Акаике (*AIC*):

$$AIC = 2\ lnL + 2k,$$

где k – число параметров модели (в случае модели *SARIMA,* $k=p + P + q + Q + d + D$), L – соответствующее значение функции правдоподобия модели.

Критерий соответствует компромиссу между точностью и сложностью для одной модели относительно нескольких других. В частности, он может быть применён для поиска оптимального набора параметров *(p,P.q,Q,d,D)* в классе моделей *ARIMA*.

Для оценки параметров модели *SARIMA* строят графики автокорреляции и частичной автокорреляции временного ряда и вычисляют значения пара-

метровследующим образом: *q* - последний несезонный лаг со значительной автокорреляцией; *p* - последнее несезонное лаг со значительной частичной автокорреляцией; *Q = Q'⁄S*, где *Q'* - последний сезонный лаг со значительнойавтокорреляцией; *P = P'⁄S*, где *P'* - последний сезонный лаг со значительнойчастичной автокорреляцией.

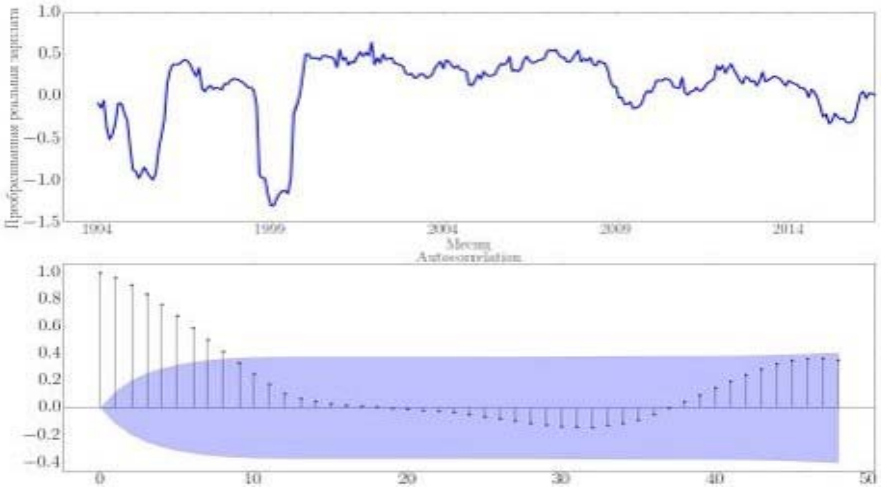
Метрики качества прогнозирования

Чаще всего для оценки качества модели предсказания временных рядовиспользуются следующие метрики:

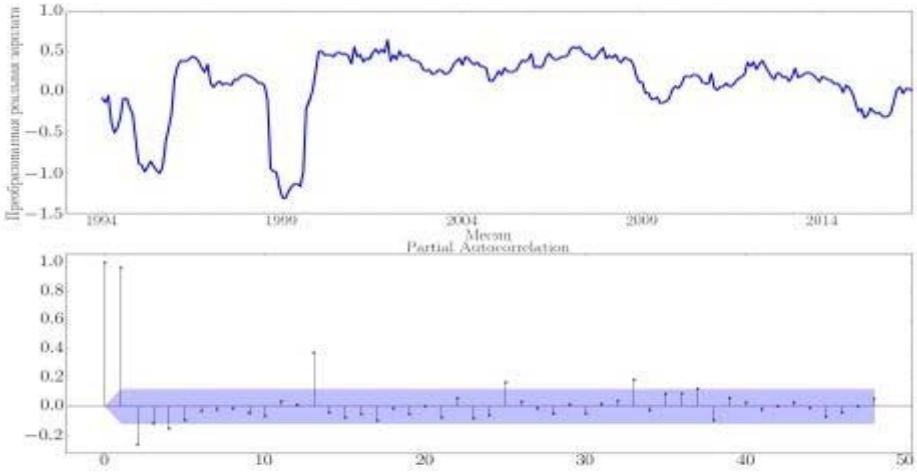
- *MAE* – Mean Absolute Error:

$$MAE = \frac{1}{n}\sum_{t=1}^{n} | y_t - \hat{y}_t |$$

а)



б)



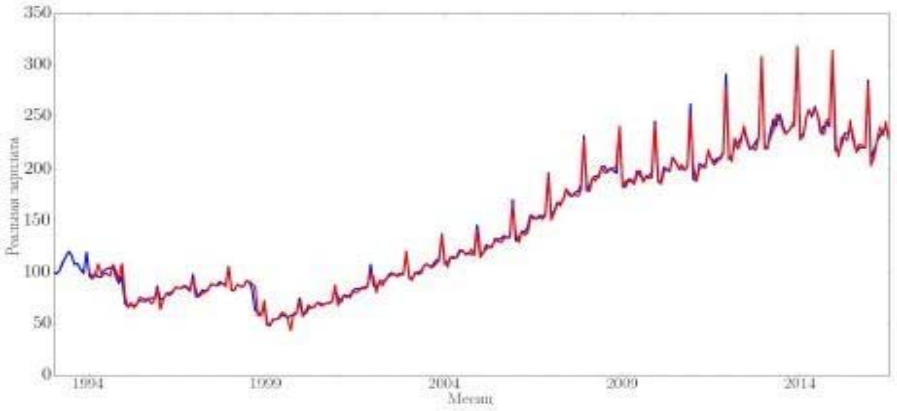
в) 

Рисунок 8.3 – (а) Коррелограмма со значениями автокорреляций временного ряда с удалённым трендом; (б) график частичных автокорреляций временногоряда; (в) исходный временной ряд (синяя кривая) и предсказания модели ARMA (2,2) (красная кривая) [3]

– *RMSE* – Root Mean Squred Error:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}$$

– *MAPE* – Mean Absolute Percentage Error:

$$MAPE = \frac{100\%}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|$$

.- *SMAPE* - Symmetric Mean Absolute Percentage Error:

$$SMAPE = \frac{100\%}{n}\sum_{t=1}^{n}\frac{|\hat{y}_t - y_t|}{\frac{1}{2} * (|y_t| + |\hat{y}_t|)}$$

где y_t - истинное значение временного ряда; $\hat{y}_t$ – предсказанное моделью значение временного ряда; n – длина рассматриваемого участка временного ряда.

Также часто вычисляют коэффициент детерминациии или R^2 («эр-квадрат»), который показывает долю объяснённой дисперсии в данных:

$$R^2 = 1 - \frac{RSS}{TSS},$$

где

$$RSS = \sum_{t=1}^{n} e_t^2 = \sum_{t=1}^{n} (y_t - \hat{y}_t)^2,$$

$$TSS = \sum_{t=1}^{n} (y_t - \bar{y}_t)^2,$$

$\bar{y}_t$ - среднее значение участка временного ряда, для которого рассчитывается метрика. Максимально достижимое значение коэффициента детерминации равно 1.0, что соответствует идеальному случаю предсказательной модели.

Задания к лабораторной работе

Задание 1. Скачайте данные с временным рядом, используя любые открытые источники.

Задание 2. Проведите тест Дики-Фуллера для проверки стационарности временного ряда.

Задание 3. Реализуйте декомпозицию временного ряда, воспользовавшись, например, средствами библиотеки statsmodels, а также дифференцирование и сезонное дифференцирование (при необходимости) средствами библиотеки pandas.

Задание 4. Проведите преобразование Бокса Кокса (при необходимости) и провести тестДики Фуллера для проверки стационарности временного ряда.

Задание 5. Обучите модель *ARIMA* с выбранным по критерию *AIC* параметрами и сделать предсказание на следующие 12 отсчётов вперёд.

Задание 6. Вычислите значения метрик *MAPE, SMAPE* и *MAE*. Для тестовой выборки (предварительно разделив временной ряд на тренировочную и тестовуючасти).

Задание 7. Сделайте вывод о качестве настроенной модели ARIMA.

Контрольные вопросы

1. Когда, по Вашему мнению, следует применять метрику MAE для оценки качества предсказания временного ряда? Приведите не менее двух примеров.

2. Каким образом можно оценить количество регрессионныхкомпонент

в модели SARIMA?

3. Сделайте предсказание временного ряда, использованного в лабораторной работе на 24 отсчёта в будущее и сравните качество предсказания в случае с горизонтом равным 12 шагам.

Обучите модель ARMA(2, 2) на использовавшемся временном ряду и сравните качество предсказания с лучшей моделью, построенной на основании критерия AIC.

Литература

1. Афанасьев В.Н., Юзбашев М.М. Анализ временных рядов ипрогнозирование — М.: Финансы и статистика, 2001. — 228 с.:
2. Портал https://machinelearningmastery.com/
3. Курс лекций «Прикладной статистический анализ данных» НИУ ВШЭhttp://wiki.cs.hse.ru/Прикладной_статистический_анализ_данных

Лабораторная работа 9

Использование Hadoop для аналитики больших данных

Цель работы: изучить основные способы использования Hadoop.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: основные способы использования Hadoop

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

Как известно парадигму MapReduce предложила компания Google в 2004 году.

Изначально Hadoop был, в первую очередь, инструментом для хранения данных и запуска MapReduce-задач, сейчас же Hadoop представляет собой большой стек технологий, так или иначе связанных с обработкой больших данных (не только при помощи MapReduce).

Основными (core) компонентами Hadoop являются:

- Hadoop Distributed File System (HDFS) – распределѐнная файловая система, позволяющая хранить информацию практически неограниченного объѐма.

- Hadoop YARN – фреймворк для управления ресурсами кластера и менеджмента задач, в том числе включает фреймворк MapReduce.

- Hadoop common.

Также существует большое количество проектов непосредственно связанных с Hadoop, но не входящих в Hadoop core:

- Hive – инструмент для SQL-like запросов над большими данными (превращает SQLзапросы в серию MapReduce–задач);

- Pig – язык программирования для анализа данных на высоком уровне. Одна строчка кода на этом языке может превратиться в последовательность MapReduce-задач;

- Hbase – колоночная база данных, реализующая парадигму BigTable;

- Cassandra – высокопроизводительная распределенная key-value база данных;

- ZooKeeper – сервис для распределѐнного хранения конфигурации и синхронизации изменений этой конфигурации;

- Mahout – библиотека и движок машинного обучения на больших данных.

Отдельно хотелось бы отметить проект Apache Spark, который представляет собой движок для распределѐнной обработки данных. Apache Spark обычно использует компоненты Hadoop, такие как HDFS и YARN для своей работы, при этом сам в последнее время стал популярнее, чем Hadoop

Установка Hadoop на кластер при помощи Clouder Manager

Раньше установка Hadoop представляла собой достаточно тяжѐлое занятие – нужно было по отдельности конфигурировать каждую машину в кластере, следить за тем, что ничего не забыто, аккуратно настраивать мониторинги. С ростом популярности Hadoop появились компании (такие как Clouder, Hortonworks, MapR), которые предоставляют собственные сборки Hadoop и мощные средства для управления Hadoop-кластером. В нашем цикле материалов мы будем пользоваться сборкой Hadoop от компании Clouder .

Для того чтобы установить Hadoop на свой кластер, нужно проделать несколько простых шагов:

1. Скачать Clouder Manager Express на одну из машин своего кластера.

2. Присвоить права на выполнение и запустить.

3. Следовать инструкциям установки.

Кластер должен работать на одной из поддерживаемых операционных систем семейства linux: RHEL, Oracle Enterprise linux, SLES, Debian, Ubuntu.

После установки вы получите консоль управления кластером, где можно смотреть установленные сервисы, добавлять/удалять сервисы, следить за состоянием кластера, редактировать конфигурацию кластера.

Запуск MapReduce программ на Hadoop

Теперь покажем как запустить MapReduce-задачу на Hadoop. В качестве задачи воспользуемся классическим примером **WordCount**,

Пример 1. Имеется набор документов. Необходимо для каждого слова, встречающегося в наборе документов, посчитать, сколько раз встречается слово в наборе.

Решение: Map разбивает документ на слова и возвращает множество пар (word, 1). Reduce суммирует вхождения каждого слова:

```
def reduce(word, values):
        def map(doc): yield word, sum(values) for
        word in doc.split():
        yield word, 1
```

Теперь задача запрограммировать это решение в виде кода, который можно будет исполнить на Hadoop и запустить.

Способ №1. Hadoop Streaming

Самый простой способ запустить MapReduce-программу на Hadoop – воспользоваться streaming интерфейсом Hadoop. Streaming-интерфейс предполагает, что map и reduce реализованы в виде программ, которые принимают данные с stdin и выдают результат на stdout.

Программа, которая исполняет функцию map называется mapper. Программа, которая выполняет reduce, называется, соответственно, reducer.

Streaming интерфейс предполагает по умолчанию, что одна входящая строка в mapper или reducer соответствует одной входящей записи для map. Вывод mapper‘а попадает на вход reducer‘у в виде пар (ключ, значение), при этом все пары соответствующие одному ключу:

- гарантированно будут обработаны одним запуском reducer‘а;

- будут поданы на вход подряд (то есть если один reducer обрабатывает несколько разных ключей – вход будет сгруппирован по ключу).

Итак, реализуем mapper и reducer на python:

```
#mapper.py
```

```
import sys
 def
do_map(doc):
for     word       in doc.split():
 yield  word.lower(), 1
for    line    in
sys.stdin:
 for  key,   value         in
do_map(line):
 print(key + “\t” + str(value))
         #reducer.py

         import sys
def
            do_re-
duce(word, values):
 return word,
sum(values)
   prev_key   =
None
values = []
for line in sys.stdin:
 key, value = line.split("“t"” if key !=
prev_key and prev_key is not None:
 result_key,  result_value  =  do_reduce(prev_key,  values)    print(re-
sult_key + "“t"”+ str(result_value)) values =  [] prev_key = key
 values.append(int(value))
 if prev_key is not
None:
```

result_key, result_value = do_reduce(prev_key, values) print(result_key + "“t"”+ str(result_value))

Данные, которые будет обрабатывать Hadoop должны храниться на HDFS. Загрузим наши статьи и положим на HDFS. Для этого нужно воспользоваться командой **hadoop fs**:

wget https://www.dropbox.com/s/opp5psid1x3jt41/lenta_articles.tar.gz tar xzvf lenta_articles.tar.gz

hadoop fs -put lenta_articles

Утилита hadoop fs поддерживает большое количество методов для манипуляций с файловой системой, многие из которых один в один повторяют стандартные утилиты linux.

Теперь запустим streaming-задачу:

yarn jar /usr/lib/h78adoopmapreduce/h78adoopstreaming.jar\ -input lenta_articles\

- output lenta_wordcount\

- file mapper.py\

- file reducer.py\

- mapper "“ython mapper.py"”

- reducer "«ython reducer.py"»

Утилита yarn служит для запуска и управления различными приложениями (в том числе mapreduce based) на кластере. Hadoop-streaming.jar – это как раз один из примеров такого yarnприложения.

Дальше идут параметры запуска:

- input – папка с исходными данными на hdfs;

- output – папка на hdfs, куда нужно положить результат;

- file – файлы, которые нужны в процессе работы map-reduce задачи;

- mapper – консольная команда, которая будет использоваться для map-стадии;

- reduce – консольная команда которая будет использоваться для reduce-стадии.

После запуска в консоли можно будет увидеть прогресс выполнения задачи и URL для просмотра более детальной информации о задаче.

Результат работы после успешного выполнения складывается на HDFS в папку, которую мы указали в поле output. Просмотреть еѐ содержание можно при помощи команды «hadoop fs -ls lenta_wordcount».

Сам результат можно получить следующим образом:

```
h79adoopfs -text lenta_wordcount/* | sort -n -k2,2 | tail -n5
с 41
что 43 на
82
и 111
в 194
```

Команда «hadoop fs -text» выдаѐт содержимое папки в текстовом виде. Я отсортировал результат по количеству вхождений слов. Как и ожидалось, самые частые слова в языке – предлоги.

Способ № 2. Сам по себе hadoop написан на java, и нативный интерфейс у hadoop-а тоже java-based. Покажем, как выглядит нативное java-приложение для wordcount:

```
import java.io.IOException; import
java.util.StringTokenizer;
 import org.apache.hadoop.conf.Configuration; import
org.apache.hadoop.fs.Path; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Job; import
org.apache.hadoop.mapreduce.Mapper; import
org.apache.hadoop.mapreduce.Reducer; import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; public class WordCount
{
```

```
public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
 private final static IntWritable one = new IntWritable(1);  private Text word = new Text();
  public void map(Object key, Text value, Context context   ) throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());  while (itr.hasMore-Tokens()) {
word.set(itr.nextToken()); context.write(word, one);   }
}
}
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable> {  private IntWritable result = new IntWritable();
  public void reduce(Text key, Iterable<IntWritable> values,   Context context
) throws IOException, InterruptedException {
int sum = 0; for (IntWritable val :
values) { sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
public static void main(String[] args) throws Exception {   Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "“ord count"”;    job.setJarByClass(Word-Count.class);
job.setMapperClass(TokenizerMapper.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
```

```
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new
Path(""dfs://localhost/user/cloudera/lenta_articles""); FileOutputFor-
mat.setOutputPath(job, new
Path(""dfs://localhost/user/cloudera/lenta_wordcount""); System.exit(job.wait-
ForCompletion(true) ? 0 : 1); } }
```

Этот класс делает абсолютно то же самое, что наш пример на Python. Мы создаѐм классы TokenizerMapper и IntSumReducer, наследуя их от классов Mapper и Reducer соответсвенно. Классы, передаваемые в качестве параметров шаблона, указывают типы входных и выходных значений. Нативный API подразумевает, что функции map на вход подаѐтся пара ключ-значение. Поскольку в нашем случае ключ пустой – в качестве типа ключа мы определяем просто Object.

В методе Main мы заводим mapreduce-задачу и определяем еѐ параметры – имя, mapper и reducer, путь в HDFS, где находятся входные данные и куда положить результат.

Для компиляции нам потребуются hadoop-овские библиотеки. Я использую для сборки Maven, для которого у cloudera есть репозиторий. В итоге файл pom.xmp (который используется maven‘ом для описания сборки проекта) у меня получился следующий):

```
<?xml version=""".0""encoding=""TF-8"">
<project xmlns=""ttp://maven.apache.org/POM/4.0.0""
xmlns:xsi=""ttp://www.w3.org/2001/XMLSchema-instance""
xsi:schemaLocation=""ttp://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd""
<modelVersion>4.0.0</modelVersion>
  <repositories>
<repository>
<id>cloudera</id>
```

<url>https://repository.cloudera.com/artifactory/clouderarepos/</url> </repository>
</repositories>
<dependencies>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>h82adoopcommon</artifactId>
<version>2.6.0-cdh5.4.2</version>
</dependency>

Соберѐм проект в jar-пакет: mvn clean package

После сборки проекта в jar-файл запуск происходит похожим образом, как и в случае streaming-интерфейса: yarn jar wordcount-1.0-SNAPSHOT.jar WordCount

Дожидаемся выполнения и проверяем результат:

с 41
что 43
на 82
и 111
в 194

hadoopfs -text lenta_wordcount/* | *sort -n -k2,2 | tail -n5*

Результат выполнения нашего нативного приложения совпадает с результатом streaming-приложения, которое мы запустили предыдущим способом.

Задание к лабораторной работе

Задание 1. Изучите примеры, предложенные в лабораторной работе.

Задание 2. Выберите произвольный текстовый документ по современным тенденциям развития информационных и реализуйте алгоритмы, предложенные в теоретической части лабораторной работы.

Контрольные вопросы

1. Что представляет собой парадигма MapReduce? Кем она была предложена?

2. Охарактеризуйте Hadoop. Перечислите его основные документы.

3. Приведите алгоритм установки Hadoop на кластер Clouder Manager.

4. Каким образом можно осуществить запуск MapReduce на Clouder Manager

Лабораторная работа 10

Работа с большими наборами данных: задачи прогнозирования

Цель: изучить принципы работы серверных приложений Map only job.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на практическом занятии: принципы работы серверных приложений Map only job.

Формируемые компетенции: ПК-2, ПК-3, ПК-8, ПК-9, ПК-12.

Теоретическая часть

MapReduce состоит из стадий Map, Shuffle и Reduce. Как правило, в практических задачах самой тяжѐлой оказывается стадия Shuffle, так как на этой стадии происходит сортировка данных. На самом деле существует ряд задач, в которых можно обойтись только стадией Map.

При создании Map-Only задачи в Hadoop нужно указать нулевое количество reducer‘ов: Map Only Job

Пример конфигурации map-only задачи на hadoop:

Native interface Hadoop Streaming Interface

Не указываем редьюсер и указываем нулевое количество редьюсеров

Пример: Указать нулевое количество редьюсеров при hadoop jar hadoop-streaming.jar \ конфигурации job‘a:

```
-D mapred.reduce.tasks=0\ job.setNumReduceTasks(0);
-input input_dir\
-output output_dir\
```

Более развѐрнутый пример:

```
-mapper "python mapper.py"\
-file "mapper.py"
```

Map Only jobs на самом деле могут быть очень полезными. Например, в платформе Facetz.DCA для выявления характеристик пользователей по их поведению используется именно один большой map-only, каждый маппер которого принимает на вход пользователя и на выход отдаѐт его характеристики.

Combine

Обычно самая тяжѐлая стадия при выполнении Map-Reduce задачи – это стадия shuffle. Происходит это потому, что промежуточные результаты (выход mapper‘а) записываются на диск, сортируются и передаются по сети. Однако существуют задачи, в которых такое поведение кажется не очень разумным. Например, в той же задаче подсчѐта слов в документах можно предварительно предагрегировать результаты выходов нескольких mapper‘ов на одном узле map-reduce задачи, и передавать на reducer уже просуммированные значения по каждой машине.

В hadoop для этого можно определить комбинирующую функцию, которая будет обрабатывать выход части mapper-ов. Комбинирующая функция очень похожа на reduce – она принимает на вход выход части mapper‘ов и выдаѐт агрегированный результат для этих mapper‘ов, поэтому очень часто reducer используют и как combiner. Важное отличие от reduce – на комбинирующую функцию попадают не все значения, соответствующие одному ключу.

Более того, hadoop не гарантирует того, что комбинирующая функция вообще будет выполнена для выхода mapper‘а. Поэтому комбинирующая функция не всегда применима, например, в случае поиска медианного значения по ключу. Тем не менее, в тех задачах, где комбинирующая функция применима, еѐ использование позволяет добиться существенного прироста к скорости выполнения MapReduce-задачи.

Использование Combiner’а на hadoop:

Native Interface Hadoop streaming

При конфигурации job-а указать класс-Combiner. Как В параметрах командной строки указать правило, он совпадает с Reducer: команду -combiner. Как правило, эта команда совпадает с командой reducer’а.

job.setMapperClass(TokenizerMapper.class);

Пример:

job.setCombinerClass(IntSumReducer.class); hadoop jar hadoop-streaming.jar job.setReducerClass(IntSumReducer.class);

```
\
-input input_dir\
-output output_dir\
-mapper "python mapper.py"\
-reducer "python reducer.py"\
-combiner "python reducer.py"\
-file "mapper.py"\
-file "reducer.py"\
```

Цепочки MapReduce-задач

Бывают ситуации, когда для решения задачи одним MapReduce не обойтись. Например, рассмотрим немного видоизмененную задачу WordCount: имеется набор текстовых документов, необходимо посчитать, сколько слов встретилось от 1 до 1000 раз в наборе, сколько слов от 1001 до 2000, сколько от 2001 до 3000 и так далее.

Для решения нам потребуется 2 MapReduce job‘a:

1. Видоизменѐнный wordcount, который для каждого слова рассчитает, в какой из интервалов оно попало.

2. MapReduce, подсчитывающий, сколько раз в выходе первого MapReduce встретился каждый из интервалов. Решение на псевдокоде:

```
#reduce2
def reduce(interval, values):
yield interval*1000, sum(values)
#map1
def map(doc):
for word in doc:
#reduce1
def reduce(word, values):
yield int(sum(values)/1000), 1
yield word, 1
```

#map2 def map(doc):

interval, cnt = doc.split()

yield interval, cnt

Для того, чтобы выполнить последовательность MapReduce-задач на hadoop, достаточно просто в качестве входных данных для второй задачи указать папку, которая была указана в качестве output для первой и запустить их по очереди.

На практике цепочки MapReduce-задач могут представлять собой достаточно сложные последовательности, в которых MapReduce-задачи могут быть подключены как последовательно, так и параллельно друг другу. Для упрощения управления такими планами выполнения задач существуют отдельные инструменты типа oozie и luigi.

Distributed cache

Важным механизмом в Hadoop является Distributed Cache. Distributed Cache позволяет добавлять файлы (например, текстовые файлы, архивы, jar-файлы) к окружению, в котором выполняется MapReduce-задача.

Можно добавлять файлы, хранящиеся на HDFS, локальные файлы (локальные для той машины, с которой выполняется запуск задачи). Я уже неявно показывал, как использовать Distributed Cache вместе с hadoop streaming: добавляя через опцию **-file** файлы mapper.py и reducer.py. На самом деле можно добавлять не только mapper.py и reducer.py, а вообще произвольные файлы, и потом пользоваться ими как будто они находятся в локальной папке.

Использование Distributed Cache:

Native API

//конфигурация Job'а

JobConf job = new JobConf();

DistributedCache.addCacheFile(new URI("/myapp/lookup.dat#lookup.dat"), job);

DistributedCache.addCacheArchive(new URI("/myapp/map.zip", job); Distribut-

edCache.addFileToClassPath(new Path("/myapp/mylib.jar"), job); Distributedcache.addCacheArchive(new URI("/myapp/mytar.tar", job); Distributedcache.addCacheArchive(new URI("/myapp/mytgz.tgz", job); *//пример использования в mapper-е:* public static
class MapClass extends MapReduceBase implements
Mapper<K, V, K, V> {
private Path[] localArchives;
private Path[] localFiles;
public void configure(JobConf job) { //
получаем кэшированные данные из архивов
File f = new File("./map.zip/some/file/in/zip.txt");
}
public void map(K key, V value,
OutputCollector<K, V> output, Reporter reporter) throws IOException {
// используем данные тут
// ... // ...
output.collect(k, v);
}
}

Hadoop Streaming

#перечисляем файлы, которые необходимо добавить в distributed cache в параметре –files. Параметр –files должен идти перед другими параметрами.
yarn hadoop-streaming.jar\
-files mapper.py,reducer.py,some_cached_data.txt\
-input '/some/input/path' \
-output '/some/output/path' \
-mapper 'python mapper.py' \
-reducer 'python reducer.py' \
пример использования:
import sys

```
#просто читаем файл из локальной папки
data = open('some_cached_data.txt').read()
for line in sys.stdin()
#processing input
#use data here
```

Reduce Join

Те, кто привык работать с реляционными базами, часто пользуются очень удобной операцией Join, позволяющей совместно обработать содержание некоторых таблиц, объединив их по некоторому ключу. При работе с большими данными такая задача тоже иногда возникает. Рассмотрим следующий пример:

Имеются логи двух web-серверов, каждый лог имеет следующий вид: \t\t.

Пример кусочка лога:

1446792139 178.78.82.1 /sphingosine/unhurrying.css
1446792139 126.31.163.222 /accentually.js
1446792139 154.164.149.83 /pyroacid/unkemptly.jpg 1446792139 202.27.13.181 /Chawia.js
1446792139 67.123.248.174 /morphographical/dismain.css 1446792139 226.74.123.135 /phanerite.php
1446792139 157.109.106.104 /bisonant.css

Необходимо посчитать для каждого IP-адреса на какой из 2-х серверов он чаще заходил. Результат должен быть представлен в виде: \t. Пример части результата:

178.78.82.1 first
126.31.163.222 second
154.164.149.83 second
226 .74.123.135 first

К сожалению, в отличие от реляционных баз данных, в общем случае

объединение двух логов по ключу (в данном случае – по IP-адресу) представляет собой достаточно тяжѐлую операцию и решается при помощи 3-х MapReduce и паттерна Reduce Join.

Общая схема ReduceJoin

ReduceJoin работает следующим образом:

1. На каждый из входных логов запускается отдельный MapReduce (Map only), преобразующий входные данные к следующему виду:

key -> (type, value)

где *key* – это ключ, по которому нужно объединять таблицы, *Type* – тип таблицы (first или second в нашем случае), а *Value* – это любые дополнительные данные, привязанные к ключу.

2. Выходы обоих MapReduce подаются на вход 3-му MapReduce, который, собственно, и выполняет объединение. Этот MapReduce содержит пустой Mapper, который просто копирует входные данные. Дальше shuffle раскладывает данные по ключам и подаѐт на вход редьюсеру в виде:

key -> [(type, value)]

Важно, что в этот момент на редьюсер попадают записи из обоих логов и при этом по полю type можно идентифицировать, из какого из двух логов попало конкретное значение. Значит данных достаточно, чтобы решить исходную задачу. В нашем случае reducere просто должен посчитать для каждого ключа записей, с каким type встретилось больше и вывести этот type.

MapJoin. Паттерн ReduceJoin описывает общий случай объединения двух логов по ключу. Однако есть частный случай, при котором задачу можно существенно упростить и ускорить. Это случай, при котором один из логов имеет размер существенно меньшего размера, чем другой. Рассмотрим следующую задачу:

Имеются 2 лога. Первый лог содержит лог web-сервера (такой же как в предыдущей задаче), второй файл (размером в 100кб) содержит соответствие URL → Тематика.

Пример 2-го файла:

/91adoop.php auto

/football/91adoop91.html sport /cars auto

/finances/money business

Для каждого IP-адреса необходимо рассчитать страницы какой категории с данного IP-адреса загружались чаще всего.

В этом случае нам тоже необходимо выполнить Join 2-х логов по URL. Однако в этом случае нам не обязательно запускать 3 MapReduce, так как второй лог полностью влезет в память. Для того, чтобы решить задачу при помощи 1-го MapReduce, мы можем загрузить второй лог в Distributed Cache, а при инициализации Mapper‘а просто считать его в память, положив его в словарь → topic. Далее задача решается следующим образом:

Map:

находим тематику каждой из страниц первого лога

input_line -> [ip, topic]

Reduce:

Ip -> [topics] -> [ip, most_popular_topic]

Reduce получает на вход ip и список всех тематик, просто вычисляет, какая из тематик встретилась чаще всего. Таким образом задача решена при помощи 1-го MapReduce, а собственно Join вообще происходит внутри map (поэтому если бы не нужна была дополнительная агрегация по ключу – можно было бы обойтись MapOnly job-ом).

Hbase

Как обычно — начнѐм с истории вопроса. Как и многие другие проекты из области BigData, Hbase зародилась из концепции которая была разработана в компании Google.

С другой стороны информацию хранящуюся в файлах довольно не-

удобно обновлять; Файлы также лишены возможности произвольного доступа. Для быстрой и удобной работы с произвольным доступом есть класс nosql-систем типа key-value storage, таких как Aerospike, Redis, Couchbase, Memcached. Однако в обычно в этих системах очень неудобна пакетная обработка данных. Hbase представляет из себя попытку объединения удобства пакетной обработки и удобства обновления и произвольного доступа.

Модель данных

Hbase — это распределенная, колоночно-ориентированная, мультиверсионная база типа «ключзначение».

Данные организованы в таблицы, проиндексированные первичным ключом, который в Hbase называется RowKey.

Для каждого RowKey ключа может храниться неограниченны набор атрибутов (или колонок).

Колонки организованны в группы колонок, называемые Column Family. Как правило в одну Column Family объединяют колонки, для которых одинаковы паттерн использования и хранения.

Для каждого аттрибута может храниться несколько различных версий. Разные версии имеют разный timestamp.

Записи физически хранятся в отсортированном по RowKey порядке. При этом данные соответствующие разным Column Family хранятся отдельно, что позволяет при необходимости читать данные только из нужного семейства колонок.

При удалении определѐнного атрибута физически он сразу не удаляется, а лишь маркируется специальным флажком tombstone. Физическое удаление данных произойдет позже, при выполнении операции Major Compaction.

Атрибуты, принадлежащие одной группе колонок и соответствующие одному ключу физически хранятся как отсортированный список. Любой атрибут может отсутствовать или присутствовать для каждого ключа, при этом если атрибут отсутствует — это не вызывает накладных расходов на хранение пустых значений.

Список и названия групп колонок фиксирован и имеет четкую схему. На уровне группы колонок задаются такие параметры как time to live (TTL) и максимальное количество хранимых версий. Если разница между timestamp для определенно версии и текущим временем больше TTL — запись помечается к удалению. Если количество версий для определѐнного атрибута превысило максимальное количество версий — запись также помечается к удалению. Модель данных Hbase можно запомнить как соответствие ключ значение:

<table, RowKey, Column Family, Column, timestamp> -> Value

Поддерживаемые операции

Список поддерживаемых операций в hbase весьма прост. Поддерживаются 4 основные операции:

- Put: добавить новую запись в hbase. Timestamp этой записи может быть задан руками, в противном случае он будет установлен автоматически как текущее время.

- Get: получить данные по определенному RowKey. Можно указать Column Family, из которой будем брать данные и количество версий которые хотим прочитать.

— Scan: читать записи по очереди. Можно указать запись с которой начинаем читать, запись до которой читать, количество записей которые необходимо считать, Column Family из которой будет производиться чтение и максимальное количество версий для каждой записи.

— Delete: пометить определенную версию к удалению. Физического удаления при этом не произойдет, оно будет отложено до следующего Major Compaction.

Архитектура

Hbase является распределенной базой данных, которая может работать

на десятках и сотнях физических серверов, обеспечивая бесперебойную работу даже при выходе из строя некоторых из них. Поэтому архитектура hbase довольна сложна по сравнению с классическими реляционными базами данных. Hbase для своей работы использует два основных процесса:

1. Region Server – обслуживает один или несколько регионов. Регион – это диапазон записей, соответствующих определенному диапазону подряд идущих RowKey. Каждый регион содержит:

- Persistent Storage – основное хранилище данных в Hbase. Данные физически хранятся на HDFS, в специальном формате Hfile. Данные в Hfile хранятся в отсортированном по RowKey порядке. Одной паре (регион, column family) соответствует как минимум один HFIle

- MemStore – буфер на запись. Так как данные хранятся в Hfile d отсортированном порядке – обновлять Hfile на каждую запись довольно дорого. Вместо этого данные при записи попадают в специальную область памяти MemStore, где накапливаются некоторое время. При наполнении MemStore до некоторого критического значения данные записываются в новый Hfile.

- BlockCache — кэш на чтение. Позволяет существенно экономить время на данных которые читаются часто.

- Write Ahead Log(WAL). Так как данные при записи попадают в memstore, существует некоторый риск потери данных из-за сбоя. Для того чтобы этого не произошло все операции перед собственно осуществление манипуляций попадают в специальный логфайл. Это позволяет восстановить данные после любого сбоя.

2. Master Server — главный сервер в кластере hbase. Master управляет распределением регионов по Region Server‘ам, ведет реестр регионов, управляет запусками регулярных задач и делает другую полезную работу. Для координации действий между сервисами Hbase использует Apache ZooKeeper, специальный сервис предназначенный для управления конфигурациями и синхронизацией сервисов.

При увеличении количества данных в регионе и достижении им определенного размера Hbase запускает split, операцию разбивающую регион на 2. Для того чтобы избежать постоянных делений регионов – можно заранее задать границы регионов и увеличить их максимальный размер.

Так как данные по одному региону могут храниться в нескольких Hfile, для ускорения работы Hbase периодически их сливает воедино. Эта операция в Hbase называется compaction. Compaction бывают двух видов:

- Minor Compaction. Запускается автоматически, выполняется в фоновом режиме. Имеет низкий приоритет по сравнению с другими операциями Hbase.

- Major Compaction**.** Запускается руками или по наступлению срабатыванию определенных триггеров(например по таймеру). Имеет высокий приоритет и может существенно замедлить работу кластера. Major Compaction лучше делать во время когда нагрузка на кластер небольшая. Во время Major Compaction также происходит физическое удаление данных, ране помеченных меткой tombstone.

Способы работы с Hbase

Hbase Shell

Самый простой способ начать работу с Hbase — воспользоваться утилитой hbase shell. Она доступна сразу после установки hbase на любой ноде кластера hbase.

Hbase shell представляет из себя jruby-консоль с встроенной поддержкой всех основных операций по работе с Hbase. Ниже приведѐн пример создания таблицы users с двумя column family, выполнения некоторых манипуляций с ней и удаление таблицы в конце на языке hbase shell:

Листинг кода:

```
create ‘users’, {NAME => ‘user_profile’, VERSIONS => 5}, {NAME => ‘user_posts’, VERSIONS => 1231231231}
put ‘users’, ‘id1’, ‘user_profile:name’, ‘alexander’ put ‘users’, ‘id1’, ‘user_profile:second_name’, ‘alexander’ get ‘users’, ‘id1’
put ‘users’, ‘id1’, ‘user_profile:second_name’, ‘kuznetsov’ get ‘users’, ‘id1’
```

get ‘users’, ‘id1’, {COLUMN => ‘user_profile:second_name’, VERSIONS => 5} put ‘users’, ‘id2’, ‘user_profile:name’, ‘vasiliy’ put ‘users’, ‘id2’, ‘user_profile:second_name’, ‘ivanov’ scan ‘users’, {COLUMN => ‘user_profile:second_name’, VERSIONS => 5} delete ‘users’, ‘id1’, ‘user_profile:second_name’ get ‘users’, ‘id1’ disable ‘users’

drop 'users'

Native Api

Как и большинство других hadoop-related проектов hbase реализован на языке java, поэтому и нативный api доступен для языке java. Вот пример использования Hbase API.

Листинг кода:

```
import
java.io.IOException;
import org.apache.hadoop.hbase.HbaseConfiguration;
import org.apache.hadoop.hbase.TableName; import
org.apache.hadoop.hbase.client.Connection; import
org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.client.Get; import
org.apache.hadoop.hbase.client.Table; import
org.apache.hadoop.hbase.client.Put; import
org.apache.hadoop.hbase.client.Result; import
org.apache.hadoop.hbase.client.ResultScanner; import
org.apache.hadoop.hbase.client.Scan; import
org.apache.hadoop.hbase.util.Bytes;
// Class that has nothing but a main.
// Does a Put, Get and a Scan against an hbase table.
// The API described here is since Hbase 1.0.
public class MyLittleHBaseClient {
 public static void main(String[] args) throws IOException { //  You need a config-
```

uration object to tell the client where to connect. // When you create a HbaseConfiguration, it reads in whatever you've set
// into your hbase-site.xml and in hbase-default.xml, as long as these can
// be found on the CLASSPATH
Configuration config = HbaseConfiguration.create();
// Next you need a Connection to the cluster. Create one. When done with it,
// close it. A try/finally is a good way to ensure it gets closed or use
// the jdk7 idiom, try-with-resources: see
//
https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResour ceClose.html
//
// Connections are heavyweight. Create one once and keep it around. From a Connection
// you get a Table instance to access Tables, an Admin instance to administer the cluster,
// and RegionLocator to find where regions are out on the cluster. As opposed to Connections,
// Table, Admin and RegionLocator instances are lightweight; create as you need them and then
// close when done.
//
Connection connection =
ConnectionFactory.createConnection(config);
try {
// The below instantiates a Table object that connects you to the "myLittleHBaseTable" table
// (TableName.valueOf turns String into a TableName instance). // When done with it, close it (Should start a try/finally after this creation so it gets
// closed for sure the jdk7 idiom, try-with-resources: see //

https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResource Close.html) Table table = connection.getTable(TableName.valueOf("myLittleHBaseTable")); try {
// To add to a row, use Put. A Put constructor takes the name of the row
// you want to insert into as a byte array. In Hbase, the Bytes class has
// utility for converting all kinds of java types to byte arrays. In the
// below, we are converting the String "myLittleRow" into a byte array to
// use as a row key for our update. Once you have a Put instance, you can
// adorn it by setting the names of columns you want to update on the row,
// the timestamp to use in your update, etc. If no timestamp, the server
// applies current time to the edits.
Put p = new Put(Bytes.toBytes("myLittleRow"));
// To set the value you'd like to update in the row 'myLittleRow', specify
// the column family, column qualifier, and value of the table cell you'd
// like to update. The column family must already exist in your table
// schema. The qualifier can be anything. All must be specified as byte
// arrays as hbase is all about byte arrays. Lets pretend the table
// 'myLittleHBaseTable' was created with a family 'myLittleFamily'.
p.add(Bytes.toBytes("myLittleFamily"), Bytes.toBytes("someQualifier"),
Bytes.toBytes("Some Value"));
// Once you've adorned your Put instance with all the updates you want to
// make, to commit it do the following (The Htable#put method takes the
// Put instance you've been building and pushes the changes you made into //
hbase)
table.put(p);
// Now, to retrieve the data we just wrote. The values that come back are
// Result instances. Generally, a Result is an object that will package up
// the hbase return into the form you find most palatable. Get g = new
Get(Bytes.toBytes("myLittleRow")); Result r = table.get(g);

```
byte [] value = r.getValue(Bytes.toBytes("myLittleFamily"), Bytes.to-
Bytes("someQualifier"));
// If we convert the value bytes, we should get back 'Some Value', the
// value we inserted at this location.
String valueStr = Bytes.toString(value);
System.out.println("GET: " + valueStr);
// Sometimes, you won't know the row you're looking for. In this case, you
// use a Scanner. This will give you cursor-like interface to the contents
// of the table. To set up a Scanner, do like you did above making a Put
// and a Get, create a Scan. Adorn it with column names, etc. Scan s = new
Scan();
s.addColumn(Bytes.toBytes("myLittleFamily"),
Bytes.toBytes("someQualifier"));
ResultScanner scanner = table.getScanner(s);
try {
// Scanners return Result instances.
// Now, for the actual iteration. One way is to use a while loop like so:
for (Result rr = scanner.next(); rr != null; rr = scanner.next()) {
// print out the row we found and the columns we were looking for
System.out.println("Found row: " + rr);
}
// The other approach is to use a foreach loop. Scanners are iterable!
// for (Result rr : scanner) {
// System.out.println("Found row: " + rr);
// }
} finally {
// Make sure you close your scanners when you are done! // That's why we have it
inside a try/finally clause
scanner.close();
}
```

```
// Close your table and cluster connection.
} finally {
                    if (table != null) table.close(); }

} finally {
        connection.close();
        }

}
}
```

Thrift, REST и поддержка других языков программирования

Для работы из других языков программирования Hbase предоставляет Thrift API и Rest API. На базе них построены клиенты для всех основных языков программирования: python, PHP, Java Script и тд.

Некоторые особенности работы с HBase

1. Hbase «из коробки» интегрируется с MapReduce, и может быть использована в качестве входных и выходных данных с помощью специальных TableInputFormat и TableOutputFormat.

2. Очень важно правильно выбрать RowKey. RowKey должен обеспечивать хорошее равномерное распределение по регионам, в противном случае есть риск возникновения так называемых «горячих регионов» — регионов которые используются гораздо чаще остальных, что приводит к неэффективному использованию ресурсов системы.

3. Если данные заливаются не единично, а сразу большими пачками – Hbase поддерживает специальный механизм BulkLoad, который позволяет заливать данные намного быстрее чем используя единичные Put‘ы. BulkLoad по сути представляет из себя двухшаговую операцию:

- формирование HFile без участия put‘ов при помощи специального MapReduce job‘a

- подкладывание этих файликов напрямую в Hbase.

4. Hbase поддерживает вывод своих метрик в сервер мониторинга Ganglia. Это может быть очень полезно при администрировании Hbase для понимания сути происходящих с hbase проблем.

Пример. Рассмотрим основную таблицу с данными, которая у нас в Data-Centric. Aliance используется для хранения информации о поведении пользователей в интернете.

RowKey

В качестве RowKey используется идентификатор пользователя, в качестве которого используется GUUID, строчка специально генерируемая таким образом, чтобы быть уникальной во всем мире. GUUID распределены равномерно, что дает хорошее распределение данных по серверам.

Column Family

В нашем хранилище используются две column family:

- **Data**. В этой группе колонок хранятся данные, которые теряют свою актуальность для рекламных целей, такие как факты посещения пользователем определенных URL. TTL на эту Column Family установлен в размере 2 месяца, ограничение по количеству версий – 2000.

- **LongData**. В этой группе колонок хранятся данные, которые не теряют свою актуальность в течение долгого времени, такие как пол, дата рождения и другие «вечные» характеристики пользователя **Колонки**

Каждый тип фактов о пользователе хранится в отдельной колонке. Например в колонке Data_v – хранятся URL, посещенные пользователем, а в колонке LongData:gender – пол пользователя.

В качестве timestamp хранится время регистрации этого факта. Например в колонке Data:_v – в качестве timestamp используется время захода пользователем на определенный URL.

Такая структура хранения пользовательских данных очень хорошо ложится на наш паттерн использования и позволяет быстро обновлять данные о

пользователях, быстро доставать всю необходимую информацию о пользователях, и, используя MapReduce, быстро обрабатывать данные о всех пользователях сразу.

Альтернативы

Hbase довольно сложна в администрировании и использовании, поэтому прежде чем использовать hbase есть смысл обратить внимание на альтернативы:

1. Реляционные базы данных. Очень неплохая альтернатива, особенно в случае когда данные влезают на одну машину. Также в первую очередь о реляционных базах данных стоит подумать в случае когда важны транзакции индексы отличные от первичного.

2. Key-Value хранилища. Такие хранилища как Redis и Aerospike лучше подходят когда необходима минимизация latency и менее важна пакетная обработка данных.

3. Файлы и их обработка при помощи MapReduce. Если данные только добавляются, и редко обновляются/изменяются, то лучше не использовать Hbase, а просто хранить данные в файлах. Для упрощения работы с файлами можно воспользоваться такими инструментами как Hive, Pig и Impala, о которых речь пойдет в следующих статьях.

Задания к лабораторной работе

Задание 1. Изучите примеры, предложенные в лабораторной работе.

Задание 2. Используя материал раздела «Теоретическая часть», реализуйте задачу прогнозирования. В качестве исходных использовать данные из лабораторной работы 9.

Контрольные вопросы

1. Перечислите основные стадии MapReduce. Дайте их характеристику.

2. Приведите алгоритм формирования цепочки MapRecude-задач.

3. Охарактеризуйте основные механизмы поиска с помощью MapReduce.

Перечень основной и дополнительной литературы

Перечень основной литературы

1. Парфенов, Ю. П. Постреляционные хранилища данных : учебное пособие для вузов / Ю. П. Парфенов ; под научной редакцией Н. В. Папуловской. — Москва : Издательство Юрайт, 2022. — 121 с.

2. Анализ данных : учебник для вузов / В. С. Мхитарян [и др.] ; под редакцией В. С. Мхитаряна. — Москва : Издательство Юрайт, 2022. — 490 с

3. Беспалов, Д. А. Администрирование баз данных и компьютерных сетей : учебное пособие / Д. А. Беспалов, А. И. Костюк ; Южный федеральный университет. – Ростов-на-Дону ; Таганрог : Южный федеральный университет, 2020. – 127 с.

Перечень дополнительной литературы:

1. Аврунев, О. Е. Модели баз данных : учебное пособие / О. Е. Аврунев, В. М. Стасышин. – Новосибирск : Новосибирский государственный технический университет, 2018. – 124 с. ЭБС

2. Бутаков, Н. А. Обработка больших данных с Apache Spark : учебно-методическое пособие : [16+] / Н. А. Бутаков, М. В. Петров, Д. Насонов. – Санкт-Петербург : Университет ИТМО, 2019. – 52 с. ЭБС