УНИВЕРСИТЕТ ИТМО

П.Е. Гладилин, К.О. Боченина

ТЕХНОЛОГИИ МАШИННОГО ОБУЧЕНИЯ

Санкт-Петербург
2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

П.Е. Гладилин, К.О. Боченина

ТЕХНОЛОГИИ МАШИННОГО ОБУЧЕНИЯ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 01.04.02, 09.04.01, 09.04.02, 09.04.03, 09.04.04, 27.04.07, 38.04.05
в качестве учебно-методического пособия для реализации основных профессиональных образовательных программ высшего образования магистратуры,

Санкт-Петербург
2020

Гладилин П.Е., Боченина К.О., Технологии машинного обучения– СПб: Университет ИТМО, 2020. – 75 с.

Рецензент(ы):
Ковальчук Сергей Валерьевич, кандидат технических наук, доцент факультета цифровых трансформаций Университета ИТМО.

В настоящем пособии представлены методические указания к выполнению лабораторных работ по дисциплине «Технологии машинного обучения».
По итогам выполнения всех работ студент должен получить теоретические знания об основных математических подходах и технологиях построения моделей машинного обучения, методах предобработки и анализа данных, основам работы с текстовыми данными, изображениями и временными рядами.

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

Содержание

Введение

В настоящем пособии представлены методические указания к выполнению лабораторных работ по дисциплине «Технологии машинного обучения» в соответствии с курсом лекций, читаемым авторами студентам Университета ИТМО, обучающимся по программам «Большие данные и машинное обучение», «Финансовые технологии больших данных», «Цифровое здравоохранение» и «Цифровые технологии умного города».

Учебно-методическое пособие разработано в соответствии с Федеральным государственным образовательным стандартом высшего профессионального образования по направлению подготовки 01.04.00 «Прикладная математика и информатика», утвержденным приказом Министерства образования и науки Российской Федерации от 28 августа 2015 г. № 911, Образовательным стандартом Университета ИТМО по направлениям подготовки 01.04.02 «Прикладная математика и информатика» и 09.04.02 «Информационные системы и технологии».

Пособие может использоваться как для самостоятельного изучения методов и алгоритмов машинного обучения студентами, аспирантами и научными работниками, так и служить руководством к решению задач, возникающих в научно-исследовательских бакалаврских, магистерских и аспирантских проектах, связанных с применением современных методов анализа данных.

Для выполнения заданий к каждой лабораторной работе необходимо скачать данные по ссылке, указанной посредством QR-кода. Задания к лабораторным работам содержат рекомендованную последовательность действий для работы с наборами данных и получения требуемого результата. В помощь к самостоятельному изучению материала в конце описания каждой лабораторной работы приведены дополнительные вопросы для самоконтроля и ссылки на дополнительную литературу.

По итогам выполнения всех восьми лабораторных работ студент должен получить теоретические знания об основных математических подходах и технологиях построения моделей машинного обучения, методах обработки и анализа данных, способах оценки качества моделей, приобрести навыки работы с современными библиотеками и фрэймворками для реализации моделей машинного обучения, навыки работы с текстовыми данными, изображениями и временными рядами.

По итогам выполнения каждой лабораторной работы студенту необходимо подготовить письменный отчет, содержащий основные результаты работы. Отчет должен быть оформлен согласно требованиям ГОСТ, включать титульный лист, основную часть и выводы. Образец титульного листа и общая структура отчёта представлены в Приложении А к учебному пособию.

Лабораторная работа № 1. Метрики качества задач классификации

Цель работы

Целью данной лабораторной работы является получение знаний основных метрик качества бинарной классификации и вариантов тонкой настройки алгоритмов классификации.

Краткие теоретические сведения

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 $[0, 1]$ (где число означает вероятность отнести пример к классу “1”). Такие пары векторов будем визуализировать так, как представлено на рисунке 1:

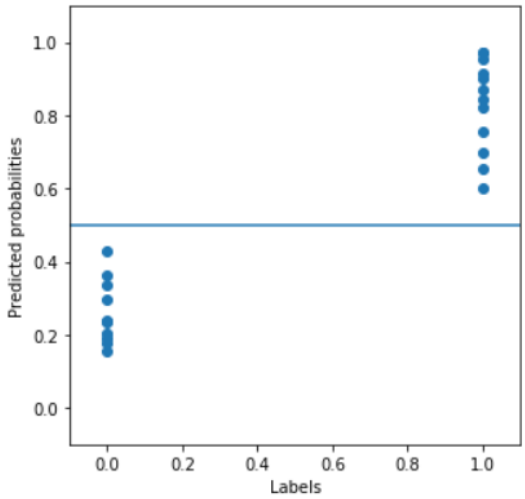


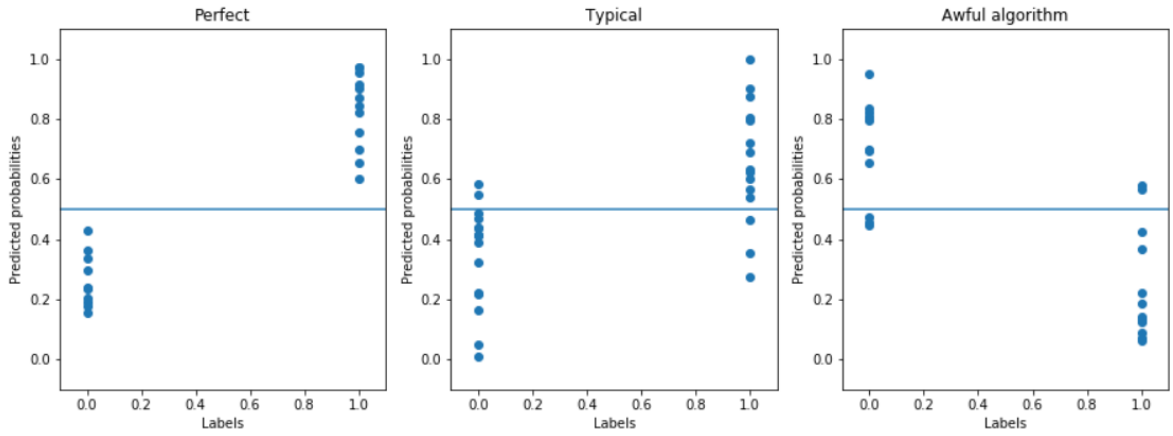
Рисунок 1 – Пример действительных (по оси абсцисс) и предсказанных (по оси ординат) значений

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

Наилучшая ситуация: порог T абсолютно верно разделяет предсказанные вероятности по двум классам. Для примера с рис. 1 идеальные интервалы вероятностей можно получить путем выбора порога $T = 0{,}5$.

Чаще всего вероятностные интервалы накладываются друг на друга (рисунок 2б) – тогда приходится подбирать порог внимательно.

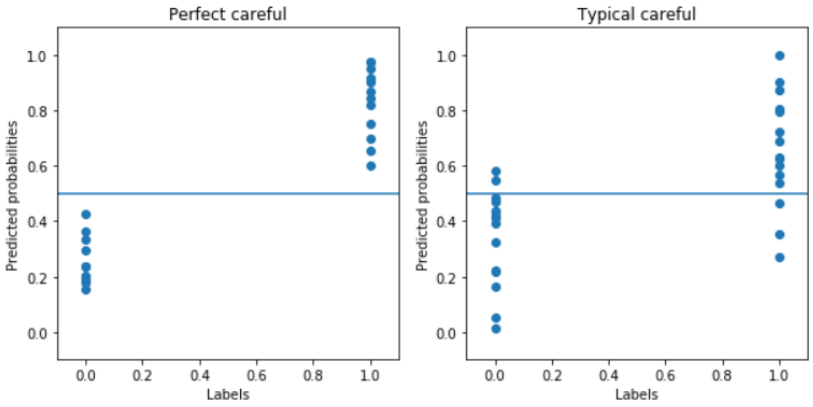
Неверно обученный алгоритм совершает обратное: он ставит вероятности объектов класса “0” выше вероятностей примеров класса “1” (рисунок 2в). В такой ситуации следует проверить, не были ли перепутаны метки “0” и “1” при получении тренировочной выборки.



а б в

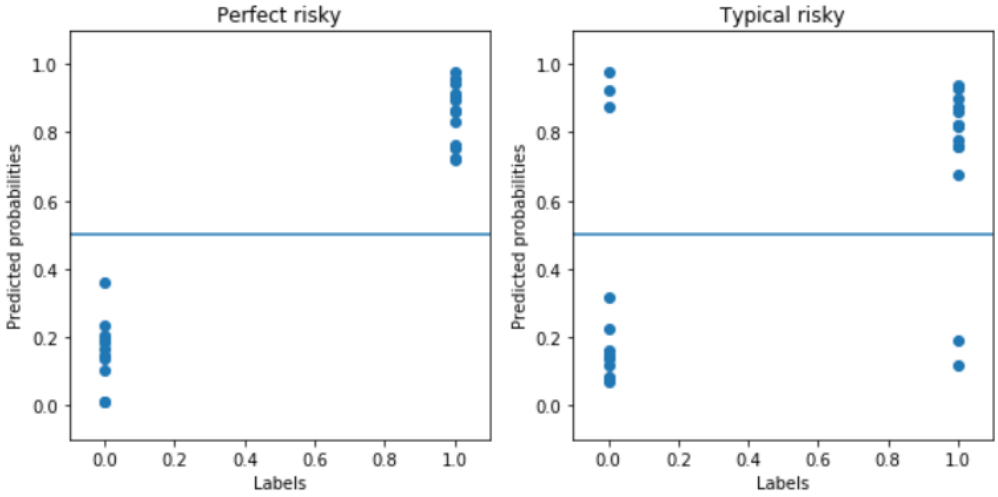
Рисунок 2 – Примеры действительного и предсказанного векторов для идеального (а), типичного (б) и неверно обученного (в) алгоритмов

Алгоритмы могут быть осторожными и выдавать значения, не слишком отдаленные от 0,5 (рис. 3), или могут брать на себя риск по получению значений, близких к нулю и единице (рис. 4).



а б

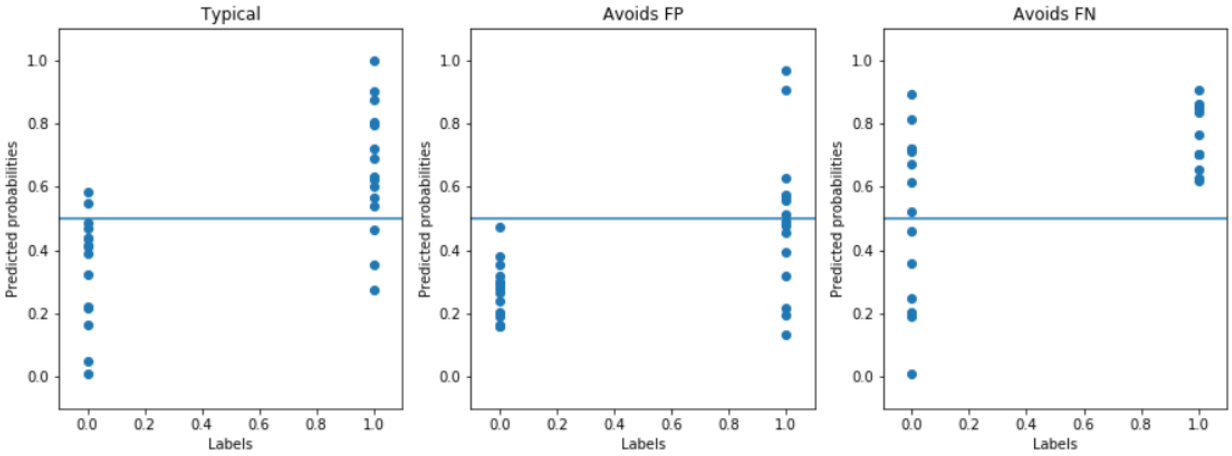
Рисунок 3 – Примеры действительного и предсказанного векторов для идеального (а) и типичного (б) осторожного алгоритма



а б

Рисунок 4 – Примеры действительного и предсказанного векторов для идеального (а) и типичного (б) рискующего алгоритма

Интервалы вероятностей могут быть смещены. Например, если нежелательны 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 выше 0,5. На рисунке 5 представлены примеры векторов в данных ситуациях.



а б в

Рисунок 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 T. 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 6 – Типы объектов: True, False – верно и неверно предсказанный класс объекта 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 7).

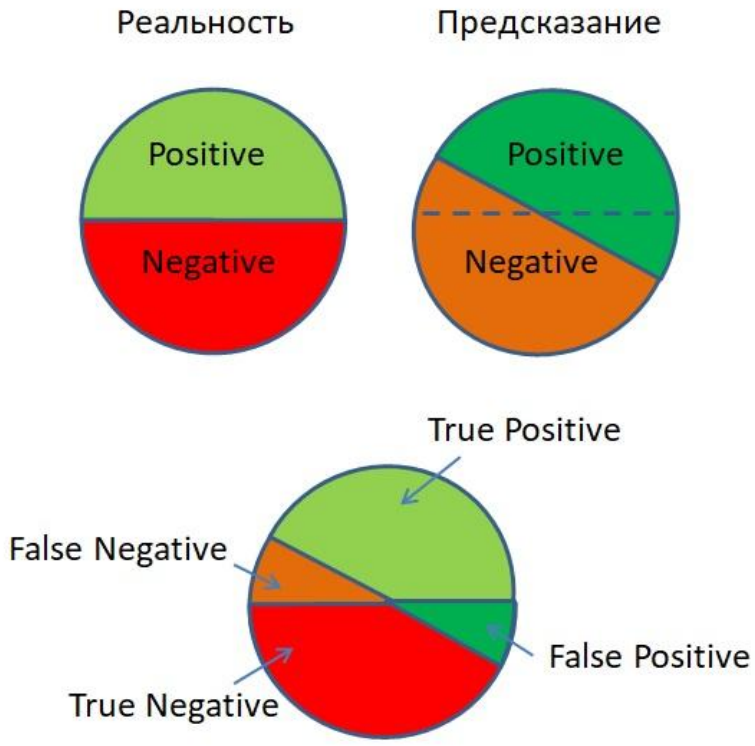


Рисунок 7 – Визуализация типов ошибок бинарной классификации

Расчёт данных метрик осуществляется по следующим формулам:

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

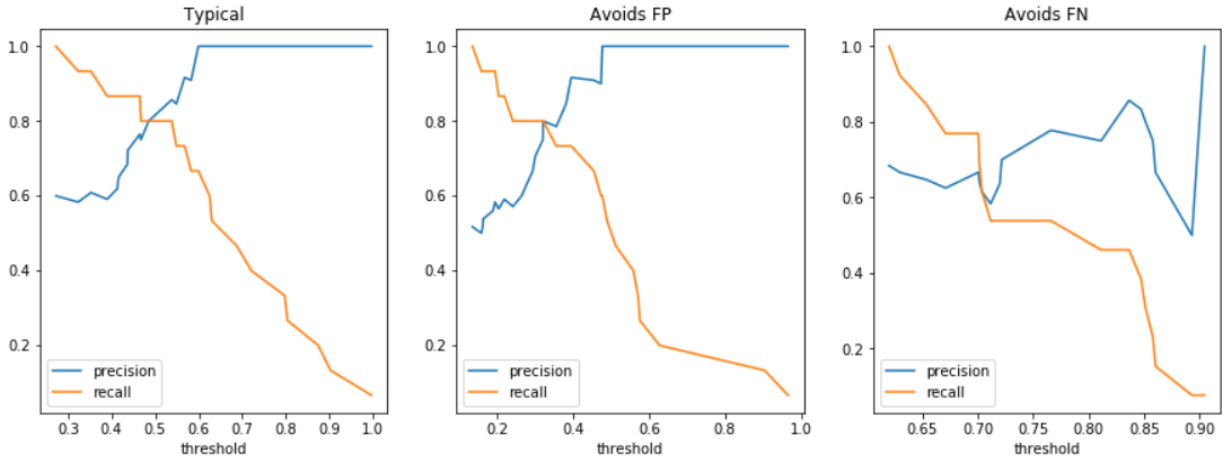
$$Recall = \frac{Tp}{Tp + Fn}.$$

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 8 представлены метрики precision и recall в зависимости от значения порога T для пар векторов с рисунка 5.



а б в

Рисунок 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. 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 5а, 5б, 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 во внимание 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 вероятностями отнесения 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 -го объекта к классу “1”, n – количество объектов. Функцию необходимо минимизировать.

Далее можно видеть значение функции ошибок для ранее использованных пар векторов.

Алгоритмы, разные по качеству (рис. 2):

Идеальный – 0,249

Типичный – 0,465

Неверно обученный – 1,527

Осторожный и рискующий алгоритмы (рис. 3 и 4):

Идеальный осторожный – 0,249

Идеальный рискующий – 0,171

Типичный осторожный – 0,465

Типичный 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. 3б и 4б, число ошибочных примеров при пороге бинаризации $T = 0{,}5$ для типичной осторожной и рискующей моделей приблизительно равно, а в случае идеальных алгоритмов (рис. 3а, 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 более обобщенной 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:

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

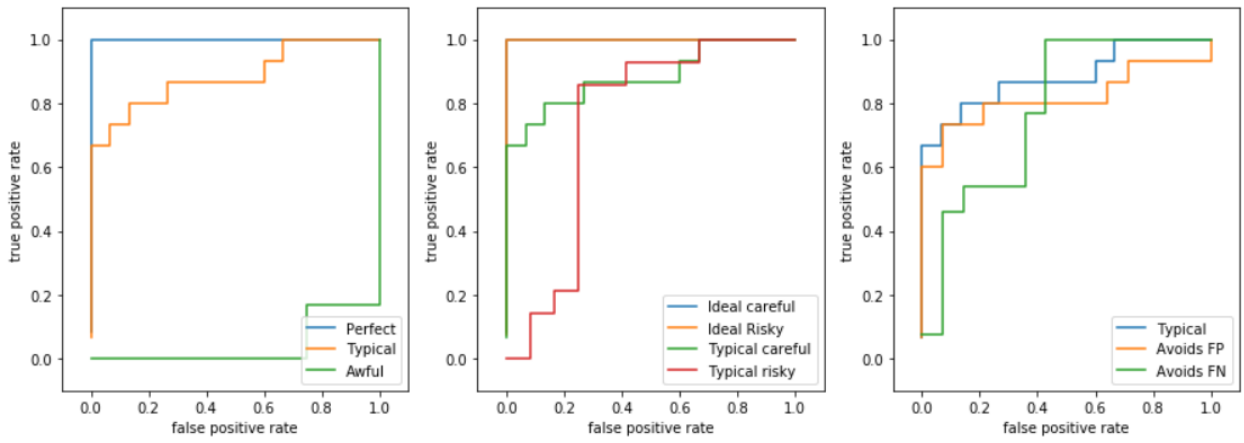
$$= -\frac{1}{n}\sum_{i=1}^{n}[0{,}3 * actual_i * \log(predicted_i) + 0{,}7 * (1 - actual_i) * \log(1 - predicted_i)].$$

Если алгоритм выдает высокое значение вероятности, а объект в действительности 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 $[0,1] \times [0,1]$. 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 9).



а б в

Рисунок 9 – ROC-кривые для пар векторов, представленных на рисунках 2 (а), 3 и 4 (б), 5 (в)

Значения AUC, соответствующие кривым на рис. 9, представлены ниже.

Алгоритмы, разные по качеству (рис. 2):

Идеальный – 1,000

Типичный – 0,884

Неверно обученный – 0,042

Осторожный и рискующий алгоритмы (рис. 3 и 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. 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. 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. 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. 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 2 классам. 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:

2) 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;

3) Рассчитайте следующие метрики, чтобы проверить качество ваших моделей:

   - доля правильных ответов (*accuracy*);

   - точность;

   - полнота;

   - $F1\text{-}score$;

   - *log-loss*.

4) Постройте *precision-recall* и ROC-кривые для ваших моделей.
5) Обучите классификатор, который избегает ошибок второго рода и рассчитайте для него метрики качества.

Дополнительные вопросы и задания

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

Целью данной лабораторной работы является получение навыков предобработки данных, необходимых для качественной настройки моделей машинного обучения.

Краткие теоретические сведения

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 заниматься предварительной 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 позволяет эффективно включать в модель категориальные признаки.

Отбор признаков. Некоторые признаки иногда не дают никакого вклада в улучшение 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 в обучении, 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 10 показан пример рассчитанных корреляций на данных для классификации мобильных телефонов по ценовым категориям[1]. Можно видеть, что наибольшее влияние на целевую функцию (price_range) оказывает размер оперативной памяти (ram), в то время как вклад признака наличия двух сим-карт (dual_sim) незначителен.

На том же наборе данных можно определить 10 атрибутов, наиболее значимых для классификации (см. рисунок 11). 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 на разных подмножествах признаков исходного тренировочного набора данных.

[1] https://www.kaggle.com/iabhishekofficial/mobile-price-classification/version/1

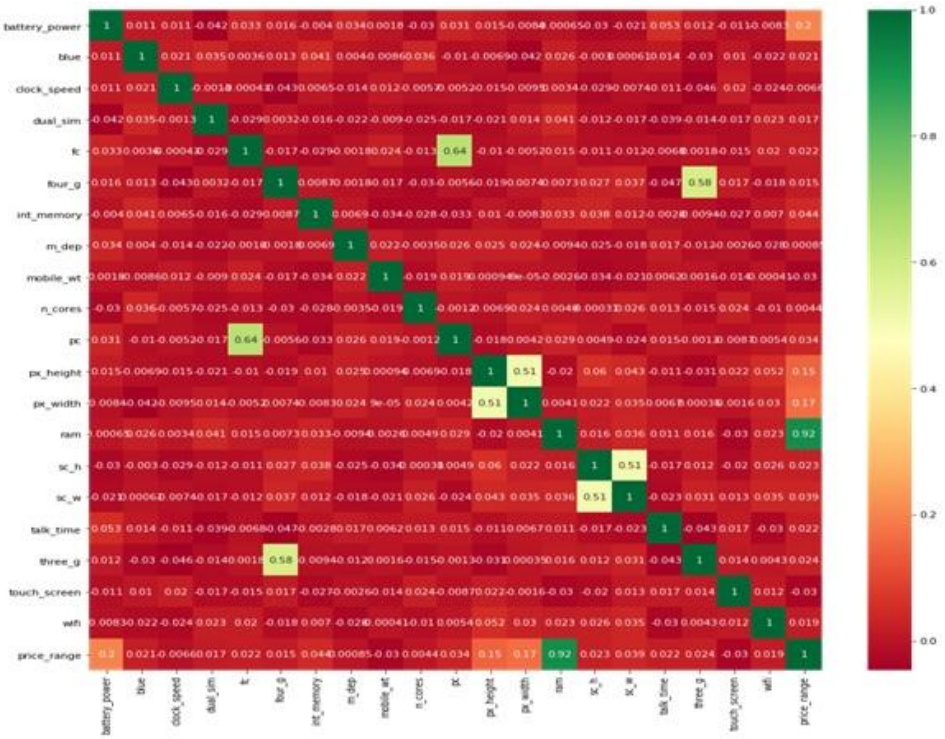


Рисунок 10 – Корреляция признаков между собой и целевой переменной

После этого выбирается подмножество признаков с наилучшими параметрами на обучающей выборке, а затем он тестируется на тестовом сете.

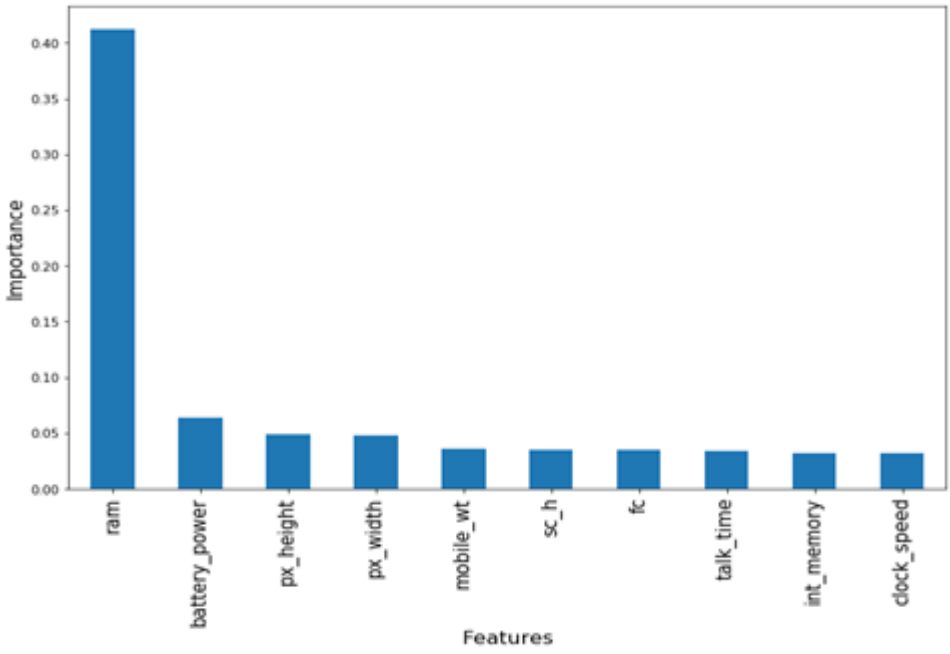


Рисунок 11 – Оценка значимости признаков по критерию $\chi 2$

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.

Задание

1) Скачать данные для исследования:

2) Реализовать функции one-hot-encoding и softmax средствами базового Python 3.6. Разделить переменные на численные и категориальные, масштабировать и нормировать данные.
3) Реализовать 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) Сделать вывод об изменении качества работы модели в зависимости от применения предобработки данных и объёма признаков, на которых обучалась модель.

Дополнительные вопросы и задания

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 зависимых признаках?

Литература

3. https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114
4. К. В. Воронцов. Отбор признаков. Лекция. http://www.machinelearning.ru/wiki/images/archive/4/4f/20111004204412%21Voron-ML-Modeling-slides.pdf

Лабораторная работа № 3. Функции ошибок в машинном обучении

Цель работы

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

Краткие теоретические сведения

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 потерь предпочтительнее, так как она не так сильно штрафует большие отклонения, нежели квадратичная функция.

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

– Quantile loss (*Lq*), функция асимметрична и больше штрафует наблюдения, оказывающиеся по нужную сторону квантили.

Графики перечисленных функций приведены на рисунке 12.

Задачи классификации

При классификации из-за принципиально другой природы распределения целевой переменной оптимизируются не сами метки классов, а их *log-*

правдоподобие. Наиболее известные варианты таких классификационных функций потерь изображены на рисунке 13.

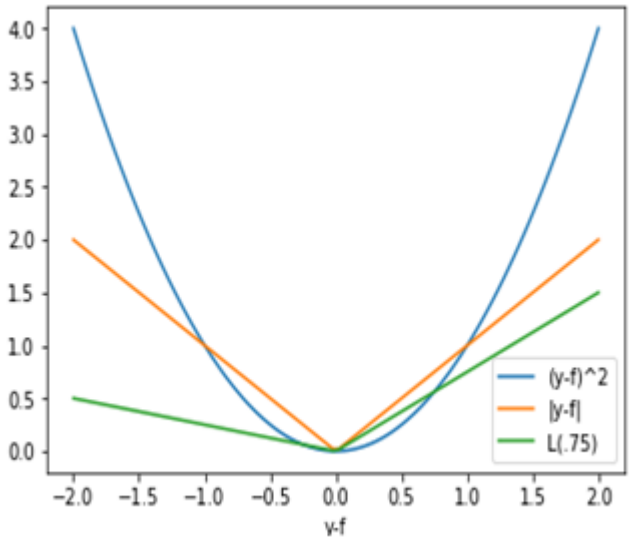


Рисунок 12 – Функции потерь для задач регрессии

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.

$$L(y, f) = \exp(-yf)$$

– Adaboost 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:

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

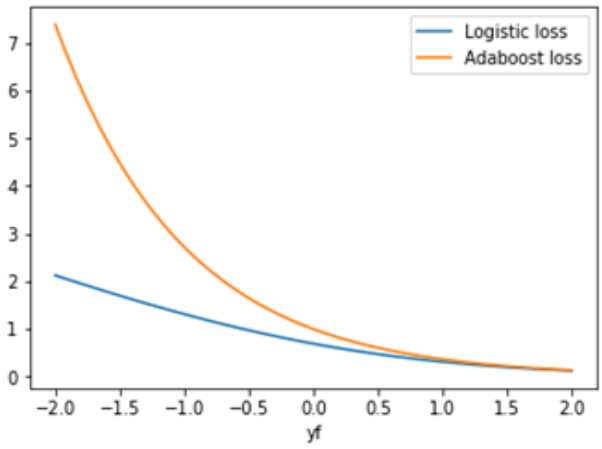


Рисунок 13 – Функции Logistic loss и Adaboost loss

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

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»:

$$CE = -\sum_{c=1}^{M} \mathrm{y}_{o,c} \cdot \log_2 p_{o,c}.$$

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.

Ход работы

1. Скачать данные:

2. Реализовать модель логистической регрессии со следующими функциями потерь:
    а) Logistic loss
    б) Adaboost loss
    в) binary crossentropy
3. Визуализировать кривые обучения модели бинарной классфикации в виде динамики изменения каждой из функций ошибок п.2 на тренировочной и тестовой выборках.

4. Сравнить качество классификации по метрике accuracy в каждом из трёх модификаций алгоритма.

Дополнительные вопросы и задания

1. Какую функцию потерь нужно применять в задаче регрессии, если вы хотите, чтобы модель больше штрафовала за выбраосы в данных?

2. Как функция кросс-энтропии связана с дивергенцией Кульбака-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 работы

Целью данной лабораторной работы является получение навыков реализации классических алгоритмов кластеризации k-means и иерархической кластеризации.

Краткие теоретические сведения

Постановка задачи кластеризации (или обучения без учителя) формулируется следущим образом. Пусть имеется обучающая выборка $X^l = \{x_1, \dots, x_l\} \subset X$ и функция расстояния между объектами ρ(x, x′), введённая в соответсвующем 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 $a : X \to Y$, 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 объектами в 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:

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

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

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

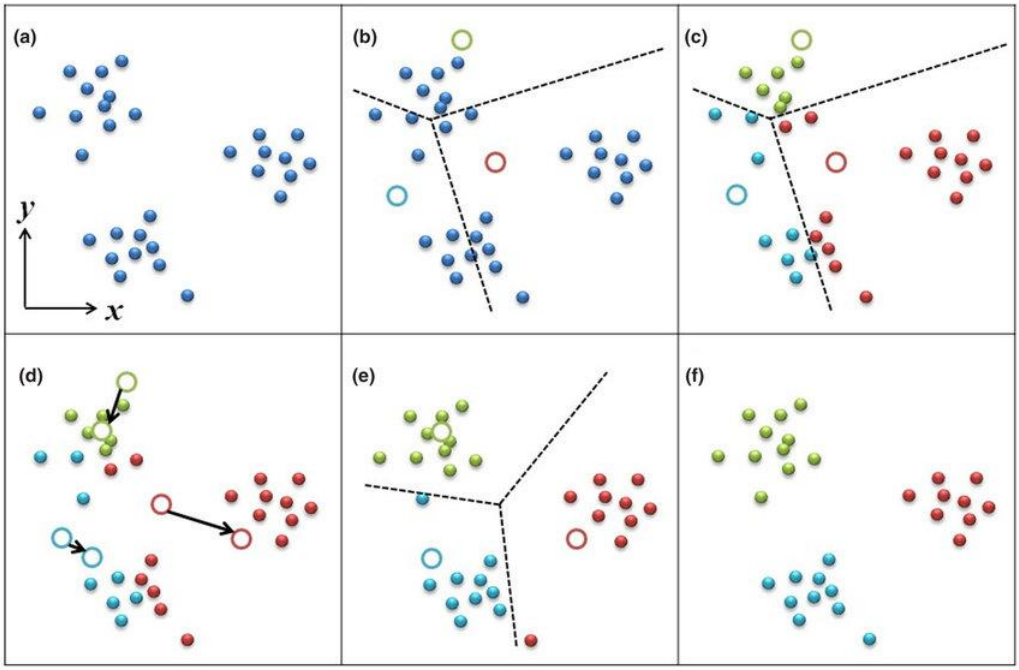


Рисунок 14– Схематическая иллюстрация выполнения последовательных итераций алгоритма *k*-means на двумерных данных с разделением на 3 кластера [1]

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 рекомендация заключается 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 более крупные, 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. Классический пример такого дерева – классификация животных и растений.

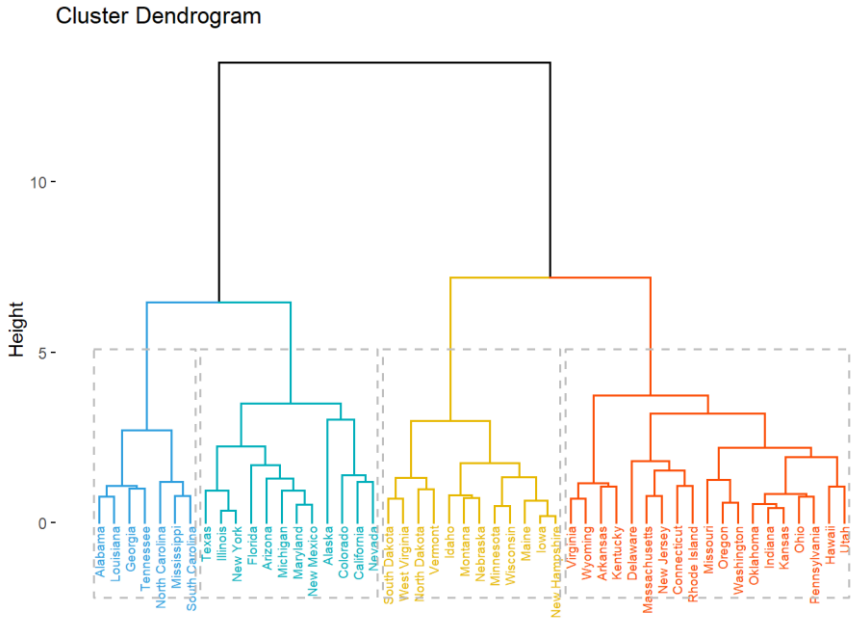


Рисунок 15 – Пример дендрограммы с последовательным объединением кластеров

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.

Ход работы

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

2) Реализовать 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;

3) Построить кривую зависимости интеркластерного расстояния от числа кластеров для алгоритма иерархической кластеризации, выбрать оптимальный порог разделения.
4) Сравнить результаты двух выбранных алгоритмов по выбранной метрики оценки качества кластеризации.

Дополнительные вопросы и задания

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 sorting strategies. Hierarchical systems // Comp. J. – 1967. – v. 9. – Pp. 373–380.

Лабораторная работа № 5. Введение в обработку естественного языка

Цель работы

Целью данной лабораторной работы (ЛР) является 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.

Краткие теоретические сведения

Задачи обработки естественного языка – одни из самых востребованных в современном 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 нинему 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 матрицу объекты-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).

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

Рисунок 16 – Пример построения матрицы частот упоминаний уникальных 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 в коллекции; $|\{d_i \in D | t \in\ d_i\}|$ - 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 17 – Пример формирования векторного представления слов в методе word2vec (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, а слова «король» и «яблоко» - далеко.

Задание

1) Скачать английскую книгу “Alice’s Adventures in Wonderland” http://www.gutenberg.org/files/11/11-0.txt

2) Реализовать 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.
3) Разделить текст на главы и в каждой главе отобрать Топ-20 слов с помощью алгоритма TF-IDF.

4) Реализовать LDA алгоритм и сравнить результаты с полученными ранее с помощью TF-IDF. Сделать выводы о применимости реализованных подходов.

Дополнительные вопросы и задания

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

2. Как бы вы назвали главы книги «Alice in Wonderland», основываясь на найденных 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 работы

Целью данной лабораторной работы является получение навыков реализации современных алгоритмов оптимизации и модификаций алгоритма градиентного спуска, широко использующегося для обучения глубоких нейронных стетей.

Краткие теоретические сведения

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 путем поиска значений -мерного вектора x;

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

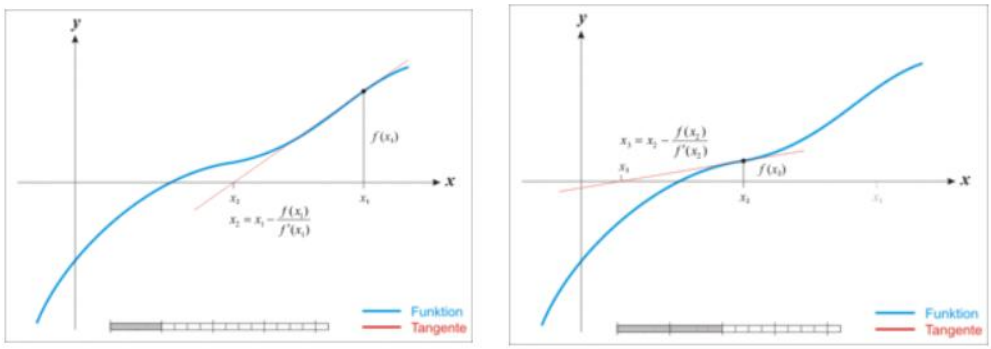
$h_j(x) = 0$ – ограничения в виде равенств;

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

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 18).



а б

Рисунок 18 – Принцип работы метода Ньютона: (а), (б) – первая и вторая итерации метода соответственно

Например, методом Ньютона после 6 итераций было найдено решение $x \approx 1{,}466$ уравнения $x^3 - x^2 - 1 = 0$ при $x_0 = 1$ (рисунок 19).

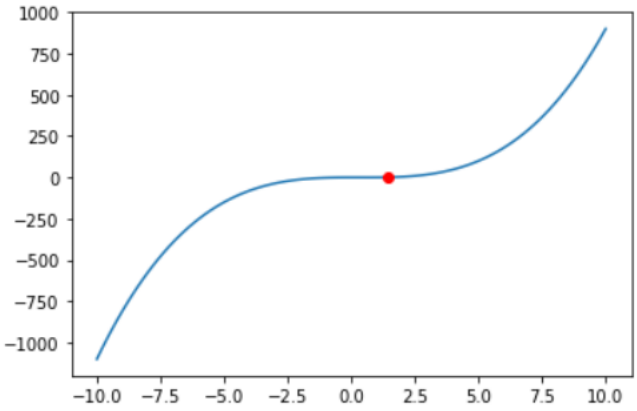


Рисунок 19 – Функция $f(x) = x^3 - x^2 - 1$ (синяя линия) и решение уравнения $f(x) = 0$ (красная точка), найденное методом Ньютона

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.

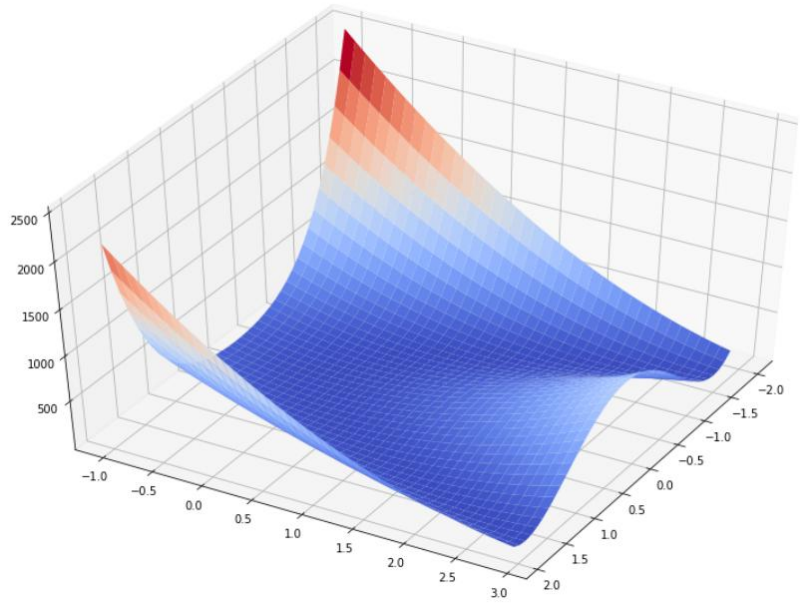
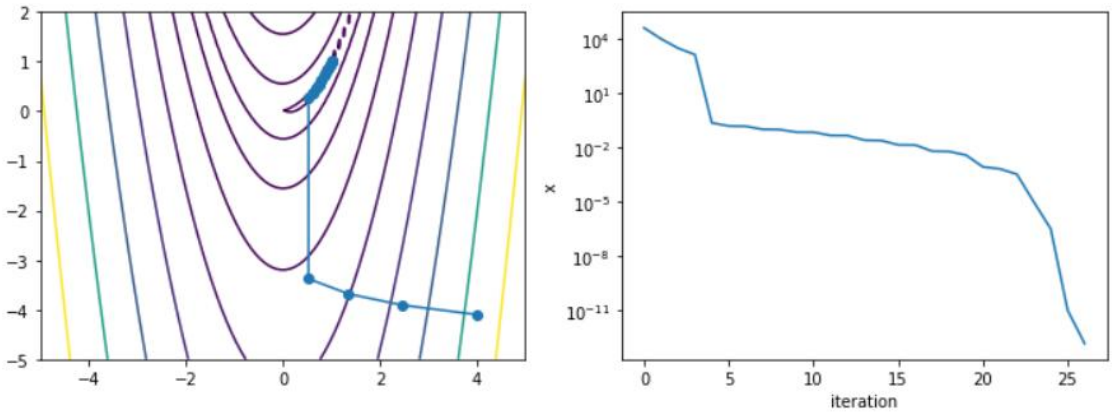


Рисунок 20 – График функции Розенброка от двух переменных

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 фунции Розенброка. Будем искать экстремум функции Розенброка из начальной точки (4; -4,1) функцией scipy.optimize.minimize() с аргументом method=’Newton-CG’ (рисунок 21).



а б

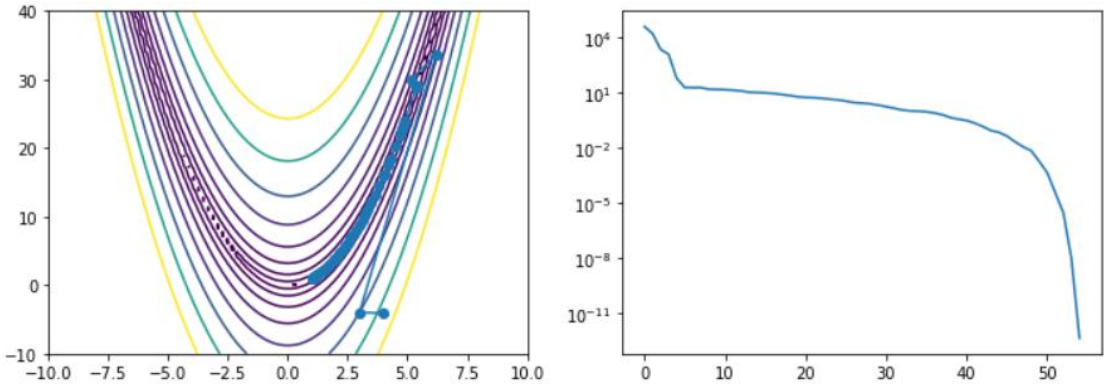
Рисунок 21– Поиск минимума функции Розенброка методом сопряженных градиентов: последовательность приближенных решений на различных итерациях алгоритма на 2D карте функции (а); график зависимости значения функции от номера итерации (б)

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 22).



а б

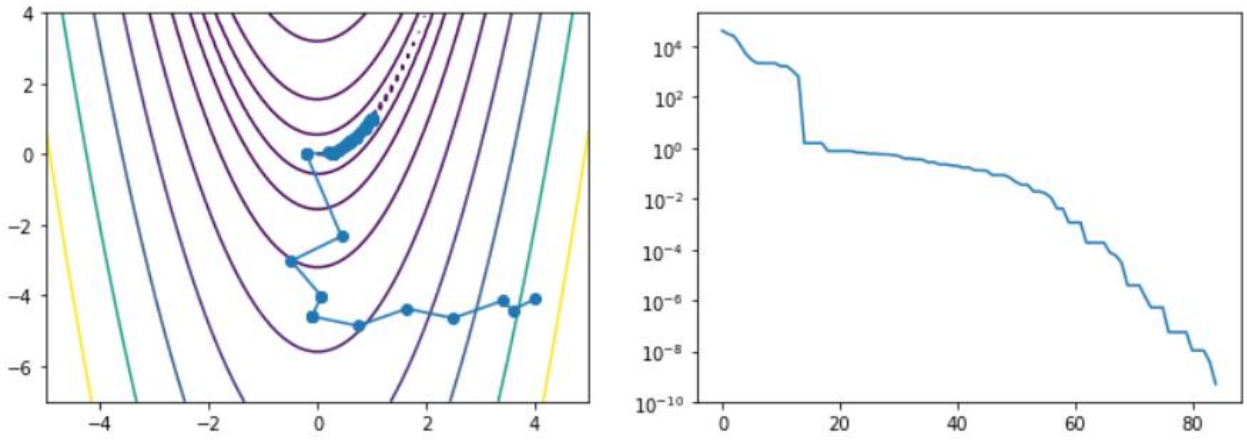
Рисунок 22 – Поиск минимума функции Розенброка алгоритмом BFGS: последовательность приближенных решений на различных итерациях алгоритма на 2D карте функции (а); график зависимости значения функции ошибки от номера итерации (б)

На 54-й итерации алгоритма было получено решение (0,99999939; 0,99999876). Из рисунка 24 (а) видно, что на втором шаге алгоритм «шагнул» в неправильном направлении, но затем последовательно начал движение к искомому минимуму функции.

Симплекс-метод Нелдера-Мида

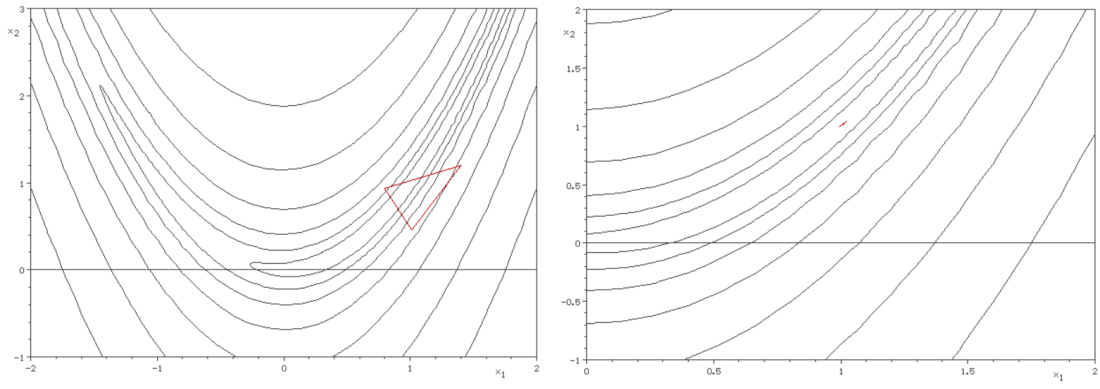
Симплекс метод, называемый также методом деформируемого многогранника, является методом безусловной оптимизации функции нескольких переменных, не использующий градиентов функции. Это свойство позволяет применять его к негладким функциям. Суть метода заключается в последовательном перемещении и деформировании 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 23 визуализирован поиск точки минимума, а на рисунке 24 – 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 23 – Поиск минимума функции Розенброка симплекс-методом Нелдера-Мида: последовательность приближенных решений на различных итерациях алгоритма на 2D карте функции (а); график зависимости значения функции ошибки от номера итерации (б)



а б

Рисунок 24 – Примеры симплексов (красные линии) на 2D карте функции Розенброка (черные линии) в начале (а) и конце (б) работы симплекс-метода Нелдера-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 зависмости 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:

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

где 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 $|x_{t+1} - x_t| < 0{,}001$.

Теперь пусть имеется дифференцируемая функция $f(x_1, x_2, \dots, x_n)$ от нескольких переменных. Наша цель – с помощью градиентного спуска найти точку минимума $(x_1^*, x_2^*, \dots, x_n^*)$. Функция имеет частные производные, хранящиеся в векторе градиента $(\frac{df(x)}{dx_1}, \frac{df(x)}{dx_2}, \dots, \frac{df(x)}{dx_n})$. Направление градиента

задает направление наибольшего подъема, а длина вектора – угол наклона. Можно легко выписать обобщенное выражение градиентного спуска для функций от нескольких переменных:

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

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: $f(x) = 0{,}1x^2 + \sin(0{,}1x^2)$ (рисунок 25).

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

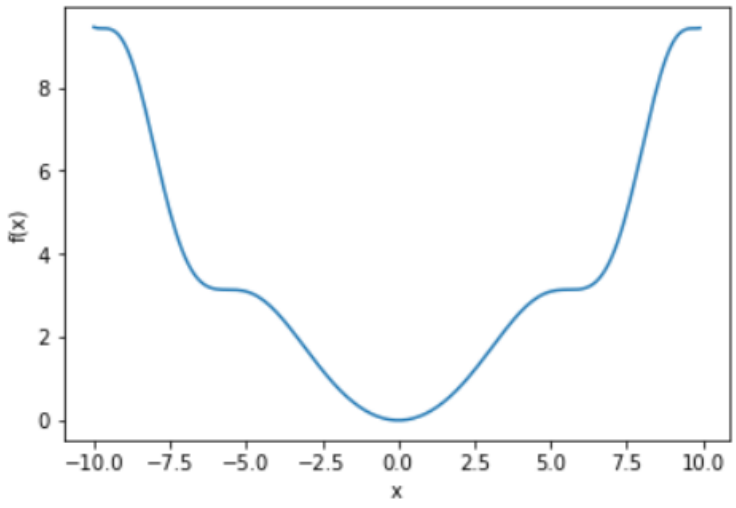


Рисунок 25 – График функции $f(x) = 0{,}1x^2 + \sin(0{,}1x^2)$

При коэффициенте скорости обучения $\lambda = 1$ и при начальной точке $x_0 = 8$ получаем схождение к минимуму функции, представленное на рисунке 26:

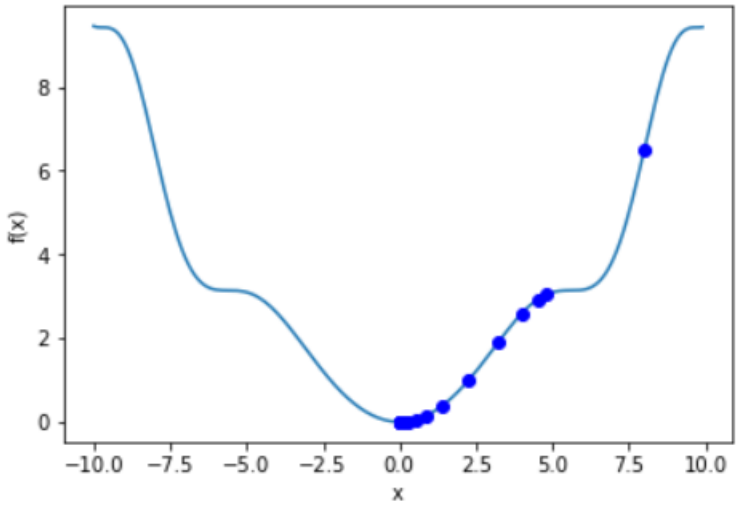


Рисунок 26 – Приближенные решения на различных итерациях градиентного спуска при поиске минимума функции $f(x) = 0{,}1x^2 + \sin(0{,}1x^2)$

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

Пример: минимизация функции нескольких переменных

Целевая функция: $f(x, y) = x^2 + y^2 + 1$ (рисунок 27), – частные производные которой $\frac{df(x)}{dx} = f_x = 2x$ и $\frac{df(y)}{dy} = f_y = 2y$.

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

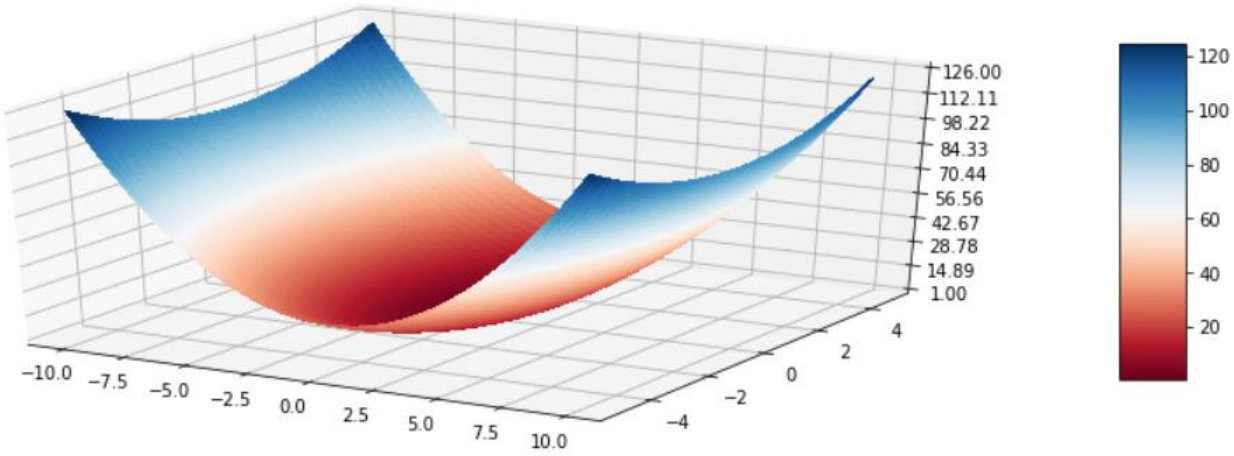


Рисунок 27 – График функции $f(x, y) = x^2 + y^2 + 1$

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 28.

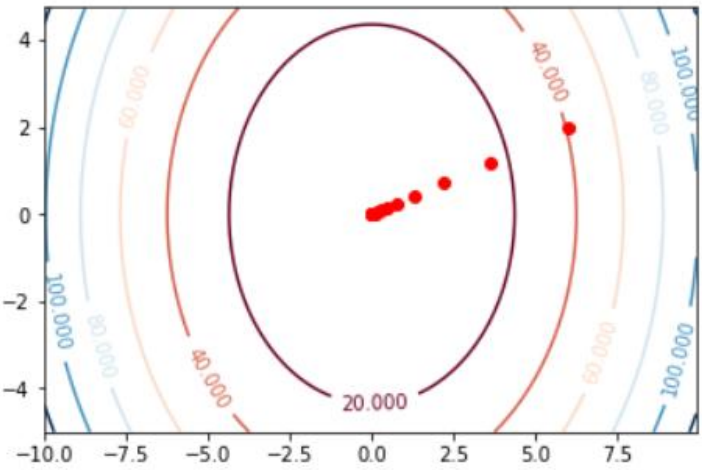


Рисунок 28 – Приближенные решения на различных итерациях градиентного спуска при поиске минимума функции $f(x) = x^2 + y^2 + 1$

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 одном шаге 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. Стохастическая модификация 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 сторону экстремума 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 параметры с 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 ε – 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:

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

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

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.

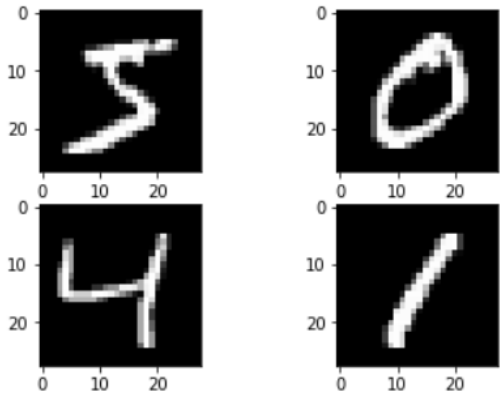
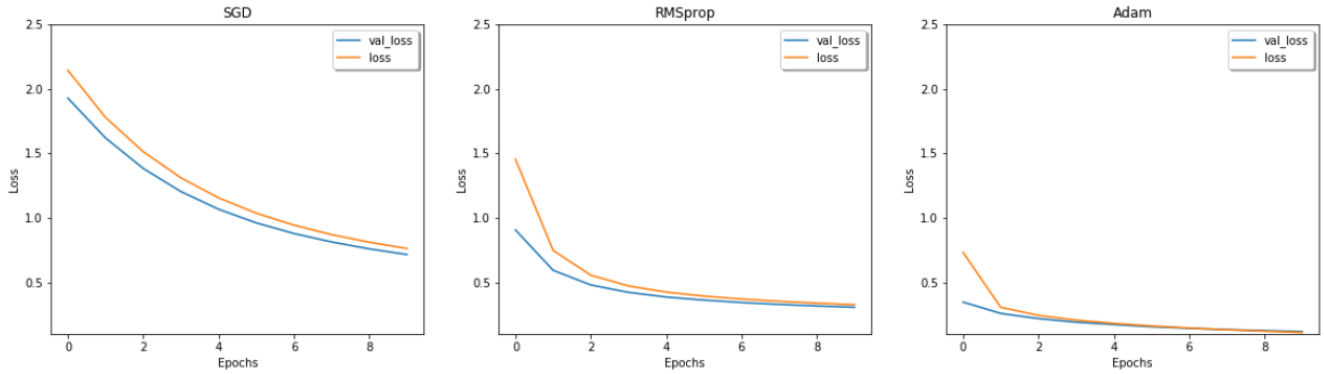


Рисунок 29 – Примеры изображений из базы данных MNIST

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 η возьмём равным $0{,}0001$. На рисунке 30 можно видеть динамику изменения функции ошибок за 10 эпох обучения модели с разными

оптимизаторами. Можно видеть преимущество использования алгоритма Adam для данной занадчи.



а б в

Рисунок 30 – Значения функции ошибок по эпохам обучения модели распознавания цифр при использовании в качестве алгоритма оптимизации: стохастического градиентного спуска (а), RMSprop (б), Adam (в)

Ход работы

1. Скачать данные

2. Реализовать функцию для алгоритма обратного распространения ошибки в виде градиентного спуска для трёхслойной полносвязной сети с 32-16-10 нейронами в слоях соответственно и среднеквадратичной функции ошибок для задачи бинарной классификации.
3. Визуализировать процесс обучения сети, построив график изменения лосс-функции.

4. Реализовать алгоритм стохастического градиентного спуска и его модификации:
    а) метод моментов;
    б) RMSprop;
    в) Адам;
5. Сделать сравнительный анализ точности решения оптимизационной задачи с помощью методов, реализованных в п.4.

Дополнительные вопросы и задания

1. Почему стохастический градиентный спуск работает быстрее, чем классический градиентный спуск?

2. Какие существют 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 Optimization Methods 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 работы

Целью данной лабораторной работы является получение навыков реализации свёрточных нейронных сетей и метода переноса обучения.

Краткие теоретические сведения

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 слой свёртки, 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 каждым фильтром в свёрточном слое, выражается в попарном перемножении элементов фильтра с элементами входного изображения, причем размер окна в точности равен размеру фильтра.

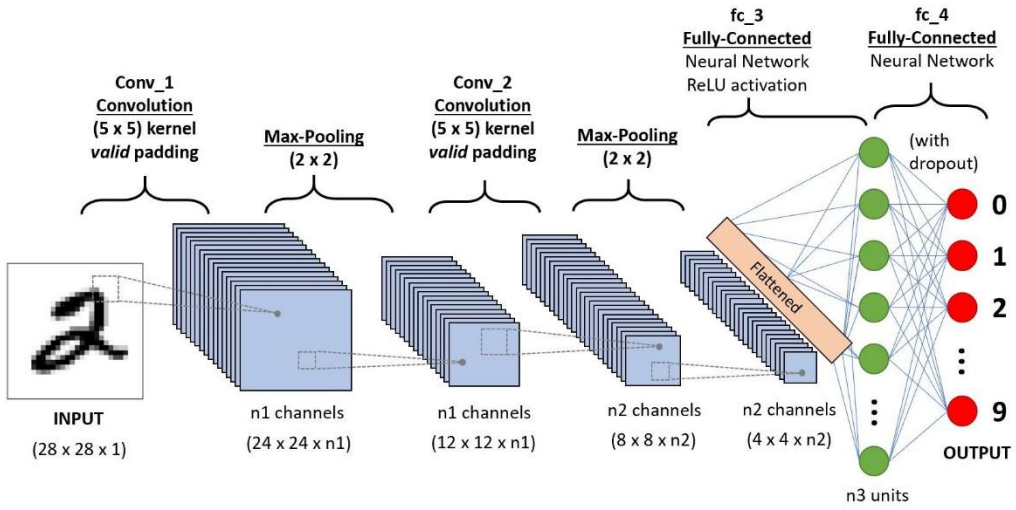


Рисунок 31 – Пример классической архитектуры сверточной нейронной сети (картинка из https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53)

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, блобов, при этом при движении к более глубоким слоям свёрточной сети фильтры в них будут выделять всё более сложные признаки (см. рисунок 34). Размер ядра обычно берут в пределах от 3x3 до 7x7. Если размер ядра маленький, то оно не сможет выделить какие-либо признаки, 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 пограничных пикселей 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.

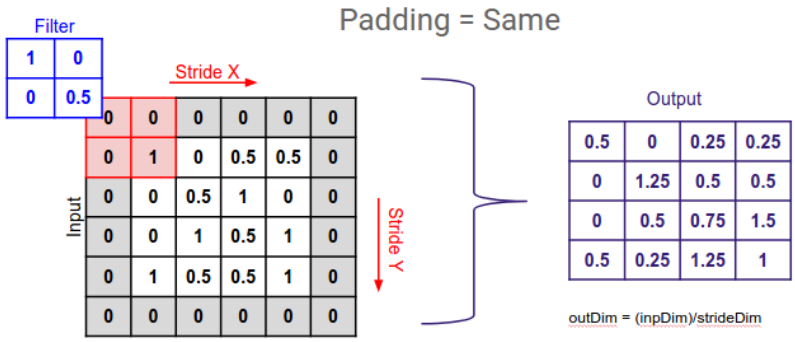


Рисунок 32 – Иллюстрация операции «padding» (изображение из https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7)

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 (см рисунок 34). Размер окна пуллинга – обычно 2x2, при этом применяют два варианта выбора значения в окне: выбор максимального элемента - “MaxPooling”) либо среднего элемента - (“AveragePooling”), которое затем записывают в соответствующую ячейку выходной матрицы.

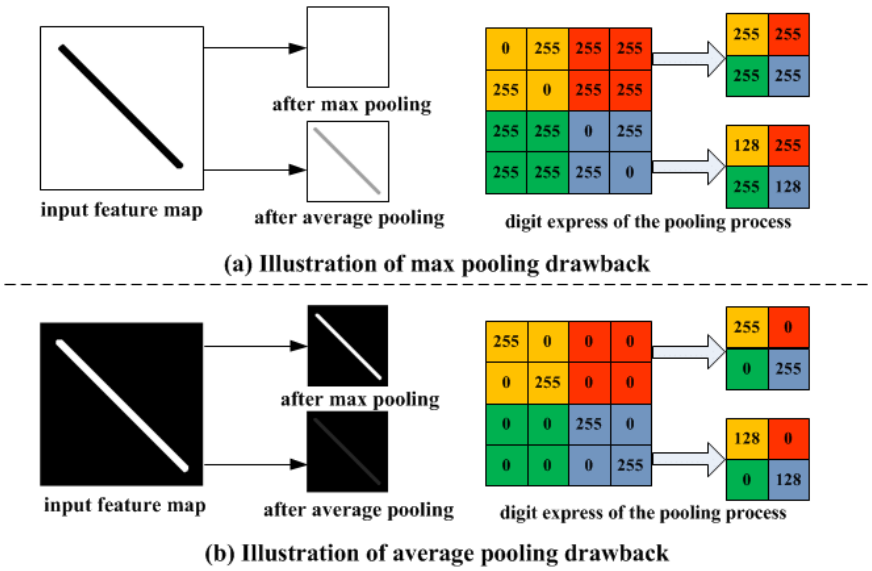


Рисунок 33 – Иллюстрация операций субдискретизации MaxPooling и AveragePooling (изображение из Yu, Dingjun & Wang, Hanli & Chen, Peiqiu & Wei, Zhihua. (2014). Mixed Pooling for Convolutional Neural Networks)

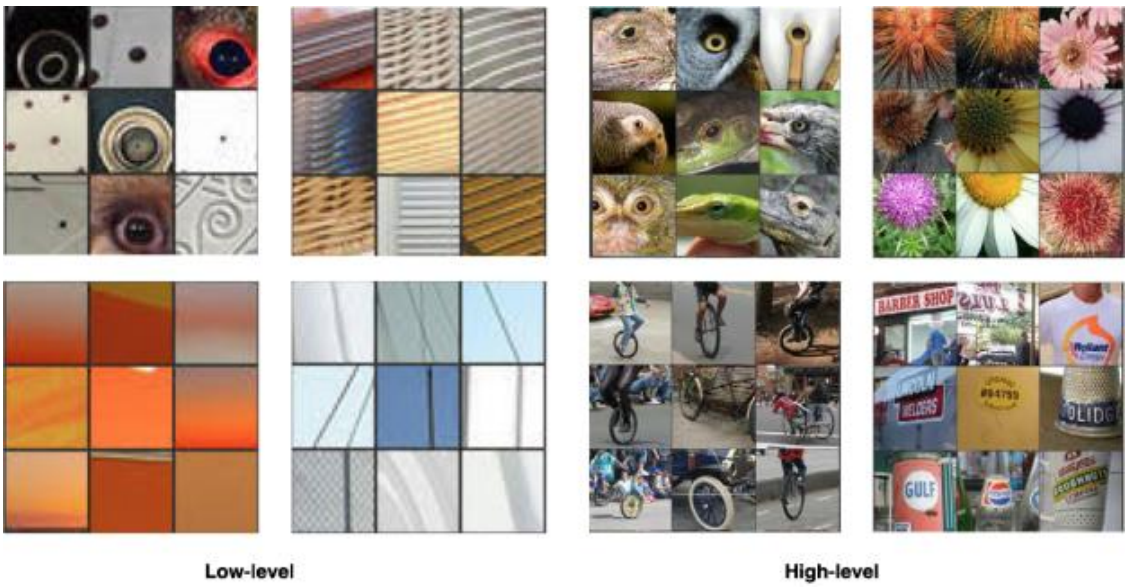


Рисунок 34 – Признаки низкого уровня (а), максимизирующие активацию нейронов на первых слоях свёрточной сети, и признаки высокого уровня (б) (максимизирующие активацию нейронов на последних слоях свёрточной сети) (картинка из https://medium.com/the-school-of-ai-official/developing-an-intuition-for-better-understanding-of-convolutional-neural-networks-17812fe4722a)

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”) и широко 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.

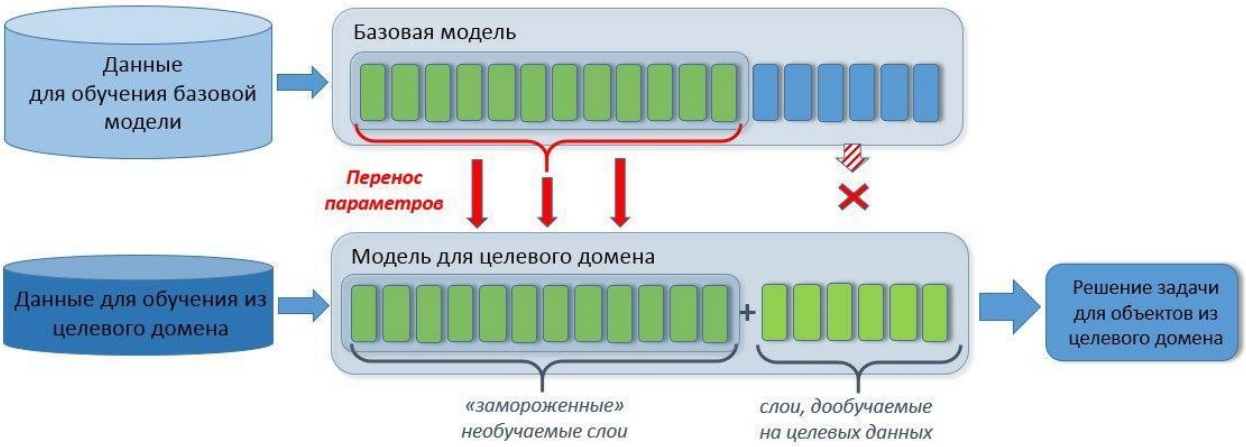


Рисунок 35 – Схема переноса обучения.

Задание

1) Скачайте данные для обучения моделей:

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.
3) реализуйте перенос обучения для моделей VGG19 и ResNet, воспользовавшись весами предобученных моделей, «заморозив» полносвязные слои и переобучив их на новых данных.
4) Сравните производительность моделей.
5) Увеличьте число эпох обучения для моделей (а) и (б) и постройте кривые обучения, демонстрирующие явление переобучения.

Дополнительные вопросы и задания

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

Целью данной лабораторной работы является получение навыков анализа временных рядов, оценки стационарности ряда и классических методов прогнозирования временных рядов.

Краткие теоретические сведения

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 36.

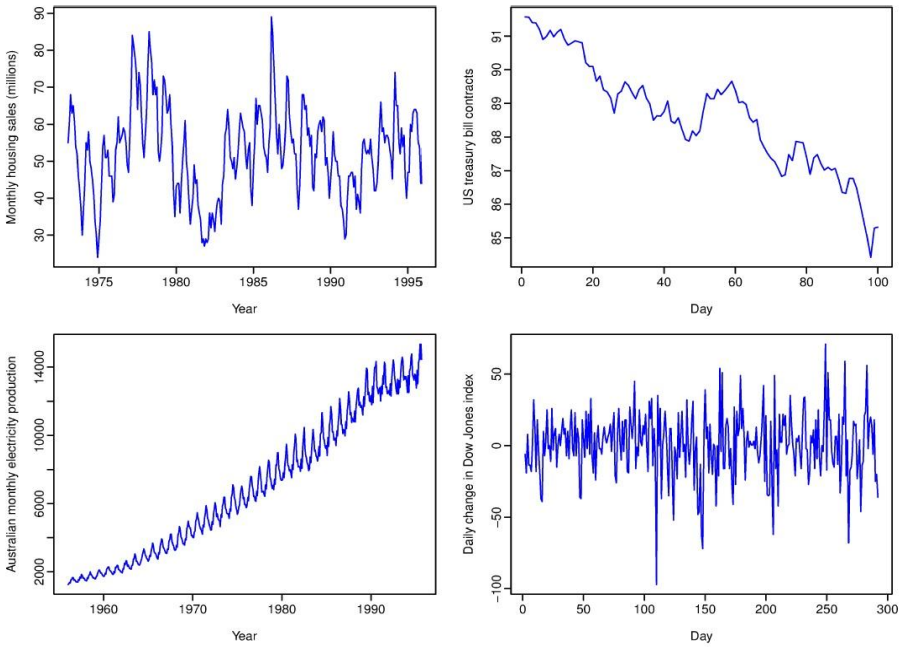


Рисунок 36 – Примеры временных рядов [3]

Важным свойством ряда также является его стационарность. Ряд $(y_1, \ldots, y_T)$ стационарен, если для любого s распределение $(y_t, \ldots, y_{t+s})$ 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 нулевой гипотезы 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 – класс одних из самых популярных классических моделей прогнозирования временных рядов. Такие модели (интегрируемые модели авторегрессии и модели скользящего среднего) – достаточно гибкие и могут описывать множество 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* слагаемых скользящего 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:

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

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

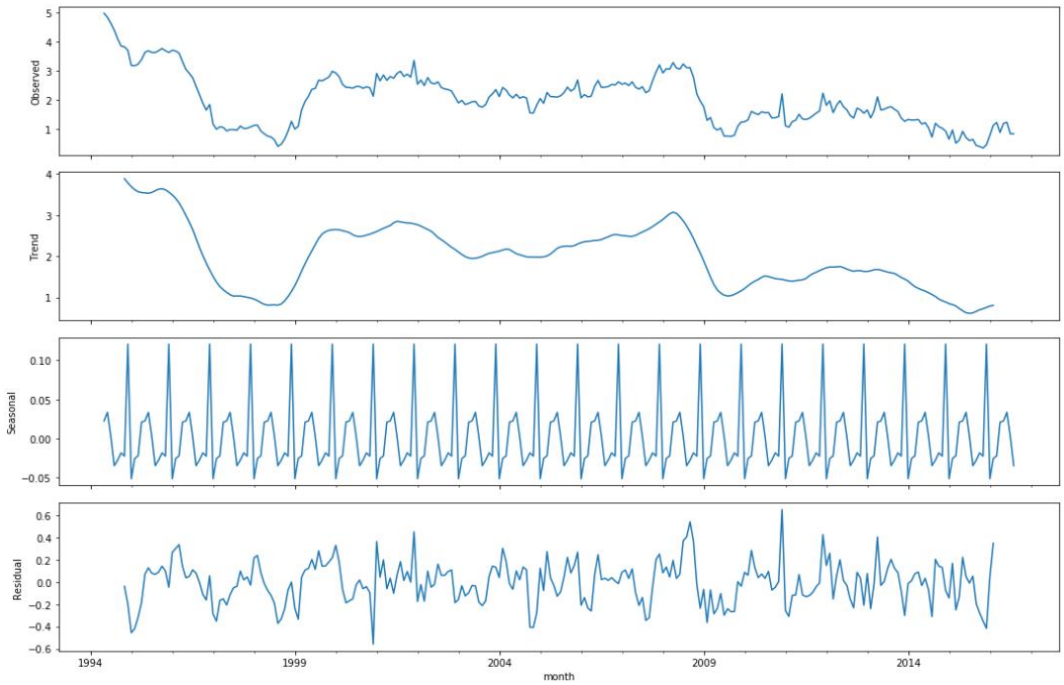


Рисунок 37 – Декомпозиция временного ряда. Сверху вниз: исходный ряд, тренд, сезонная компонента, остатки (шум) [3]

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 автккорреляции временного ряда и вычисляют значения параметров следующим образом (см. рисунок 39):

- q - последний несезонный лаг со значительной автокорреляцией:
- p - последнее несезонное лаг со значительной частичной автокорреляцией;
- $Q = Q'/S$, где Q' - последний сезонный лаг со значительной автокорреляцией;
- $P = P'/S$, где P' - последний сезонный лаг со значительной частичной автокорреляцией.

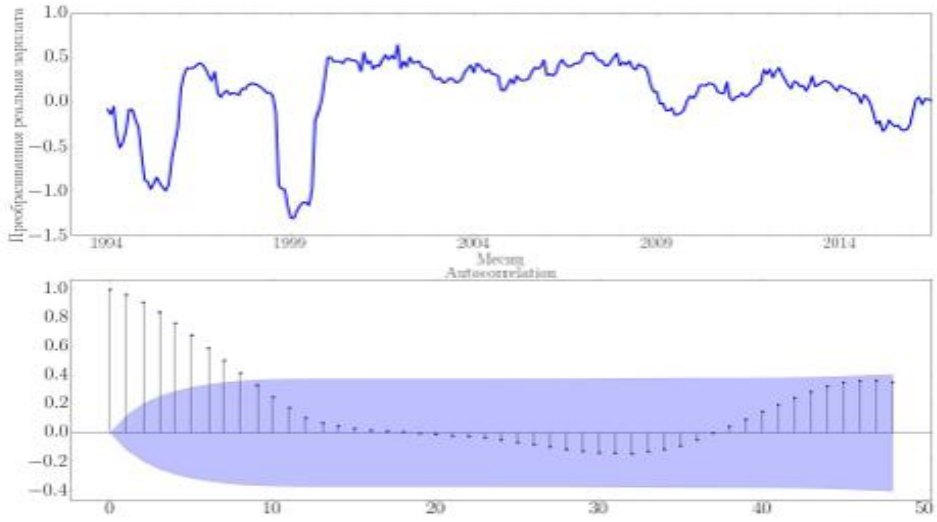
Метрики качества прогнозирования

Чаще всего для оценки качества модели предсказания временных рядов используются следующие метрики:

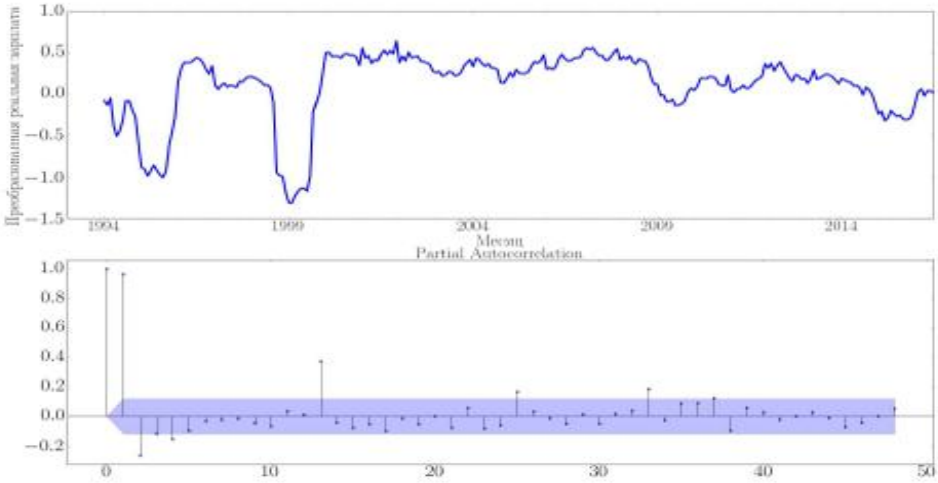
- *MAE* – Mean Absolute Error:

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

а)



б)



в)

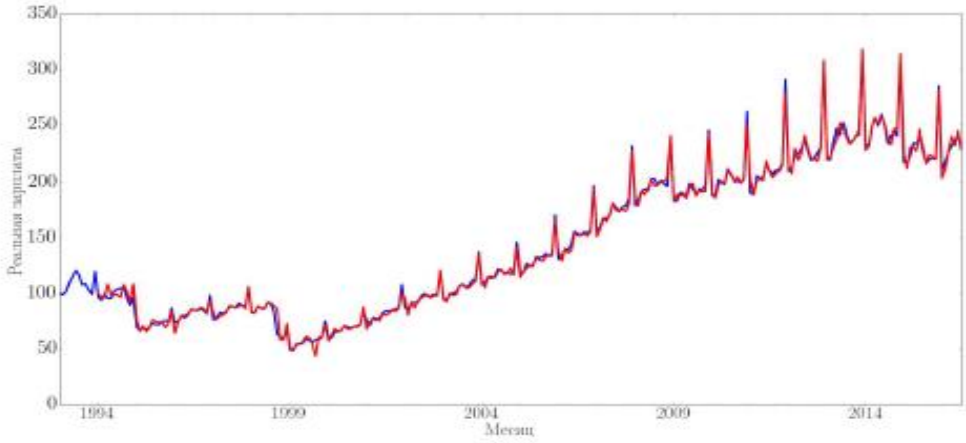


Рисунок 38 – (а) Коррелограмма со значениями автокорреляций временного ряда с удалённым трендом; (б) график частичных автокорреляций временного ряда; (в) исходный временной ряд (синяя кривая) и предсказания модели ARMA (2,2) (красная кривая) [3]

– *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

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

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

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

Ход работы

1) Скачать данные с временным рядом:

2) Провести тест Дики-Фуллера для проверки стационарности временного ряда;
3) Реализовать декомпозицию временного ряда, воспользовавшись, например, средствами библиотеки statsmodels, а также дифференцирование и сезонное дифференцирование (при необходимости) средствами библиотеки pandas.
4) Провести преобразование Бокса Кокса (при необходимости) и провести тест Дики Фуллера для проверки стационарности временного ряда.
5) Обучить модель *ARIMA* с выбранным по критерию *AIC* параметрами и сделать предсказание на следующие 12 отсчётов вперёд.
6) Вычислить значения метрик *MAPE*, *SMAPE* и *MAE*. Для тестовой выборки (предварительно разделив временной ряд на тренировочную и тестовую части).
7) Сделать вывод о качестве настроенной модели ARIMA.

Дополнительные вопросы и задания

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 регрессионных компонент 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.*
4) *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 и прогнозирование — М.: Финансы и статистика, 2001. — 228 с.:

2. Портал https://machinelearningmastery.com/

3. Курс лекций «Прикладной статистический анализ данных» НИУ ВШЭ http://wiki.cs.hse.ru/Прикладной_статистический_анализ_данных

Приложение А

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

о выполнении лабораторной работы № X
«Тема работы»

Работу выполнил:
ст. группы <Номер группы> Фамилия И.О.
Работу принял:
Фамилия И.О.

Санкт-Петербург
2020

Цель работы

Какая цель преследуется при выполнении лабораторной работы (0.1 – 0.2 стр.).

Постановка задачи

Задача, которая решается при выполнении этой лабораторной работы (0.2 – 0.3 стр.).

Краткая теоретическая часть

Краткие теоретические сведения о теме, по которой выполняется лабораторная работа. Основы используемых методов и алгоритмов: свойства, достоинства, недостатки (не более 1 стр.).

Результаты

Представление результатов (промежуточные и итоговые выкладки, графики), краткое обсуждение результатов, оценка качества алгоритмов (2–3 стр.).

Заключение

Выводы на основе достигнутых результатов, оценка возможности применения приобретённых навыков и умений на практике (0.2 – 0.3 стр.).

Гладилин Петр Евгеньевич
Боченина Клавдия Олеговна

Технологии машинного обучения

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверский пр., 49