

Урок 1

Прогнозирование временных рядов

1.1. Временные ряды.

Этот урок посвящён классическим задачам, связанным с временными рядами. Задачи такого типа часто возникают в бизнес-аналитике.

Временным рядом называется последовательность значений признака y , измеряемого через постоянные временные интервалы:

$$y_1, \dots, y_T, \dots, y_t \in \mathbb{R}.$$

В этом определении нужно обратить внимание на то, что временные интервалы между измерениями признака постоянны.

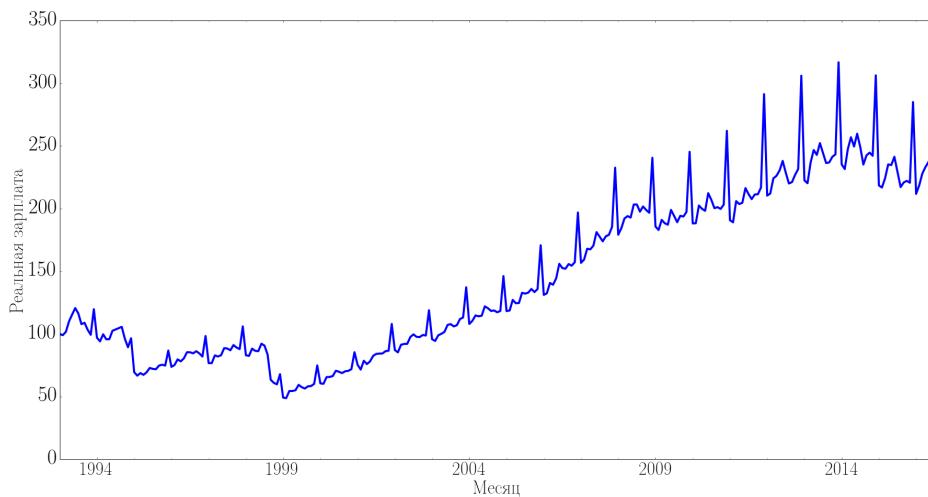


Рис. 1.1: Среднемесячная реальная заработная плата в России, выраженная в процентах от её значения в январе 1993 г.

Примеры временных рядов — это ряды среднедневных цен на акции определённой компании, среднемесячного уровня безработицы, измеренного в течение нескольких лет, среднегодового уровня производства автомобилей. Ещё один пример временного ряда (показан на рисунке 1.1) — это реальная заработная плата в России, выраженная в процентах от её значения на январь 1993 г., измеренная и усреднённая за каждый месяц, начиная с того момента.

1.1.1. Прогнозирование временного ряда.

Интерес представляет задача прогнозирования временных рядов. Подразумевается, что зная значение признака в прошлом, можно предсказать его в будущем. Формально задача ставится как поиск функции f_T :

$$y_{T+d} \approx f_T(y_T, \dots, y_1, d) \equiv \hat{y}_{T+d|T},$$

где $d \in \{1, \dots, D\}$ — отсрочка прогноза, D — горизонт прогнозирования.

1.1.2. Главная особенность временных рядов

До этого, на протяжении практически всей специализации, считалось, что анализируемые данные — это простые выборки, то есть независимые одинаково распределённые наблюдения. В задаче анализа временных рядов всё с точностью наоборот: предполагается, что данные в прошлом каким-то образом связаны с данными в будущем. Чем сильнее они связаны, тем больше имеется информации о поведении временного ряда в будущем и тем точнее можно сделать прогноз.

Полезно снова рассмотреть данные о реальной среднемесячной зарплате в России (рис. 1.1). Видно, что на графике изображена не простая выборка (измерения не являются независимыми и одинаково распределёнными), а сложный, структурированный процесс. Выявив структуру этого процесса, можно учесть её в прогнозирующей модели и построить действительно точный прогноз.

1.1.3. Применение модели регрессии

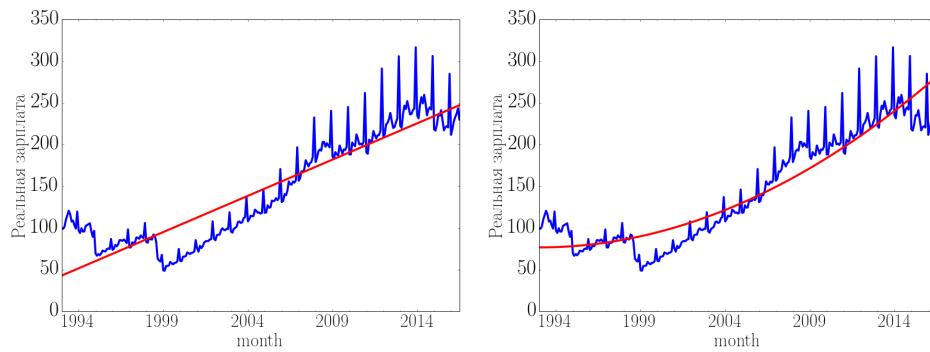


Рис. 1.2: Применение модели линейной (слева) и квадратичной (справа) регрессии к задаче прогнозирования временного ряда.

До этого в курсе большое внимание было уделено задаче обучения с учителем. Можно попробовать свести к ней задачу прогнозирования временного ряда. Процесс разворачивается во времени, поэтому кажется логичным задать признаки, связанные со временем и попробовать решить задачу, применяя модель регрессии. Регрессия может быть линейной или, например, квадратичной (рис. 1.2).

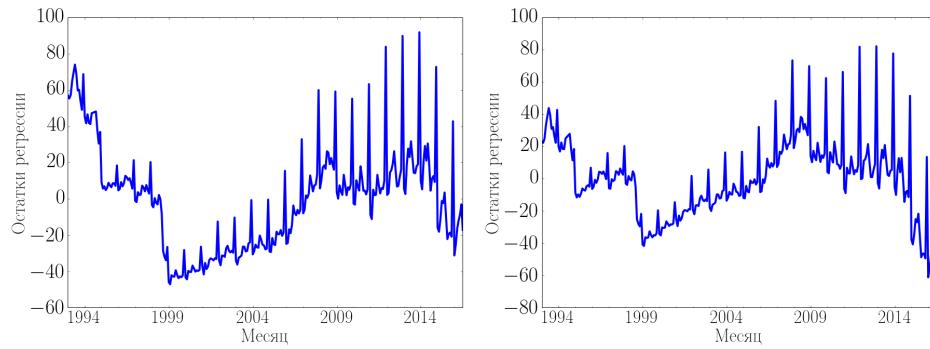


Рис. 1.3: Остатки модели линейной (слева) и квадратичной (справа) регрессии в задаче прогнозирования временного ряда.

Однако это решение слишком простое, чтобы быть хорошим. Остатки такой регрессии (рис. 1.3) далеко не похожи на случайный шум, в них остаётся большая часть структуры, которая не была учтена в регрессионной модели. Чем больше структуры временного ряда учитывается в модели, тем лучшее предсказание она даёт. Вид остатков регрессии намекает на то, что можно построить более сложную модель, которая будет лучше описывать имеющиеся данные, а также давать более точные прогнозы в будущем. Построению таких моделей будет посвящена оставшаяся часть урока.

1.1.4. Компоненты временных рядов

Полезно рассмотреть несколько понятий, которыми можно описать поведение временных рядов:

- Тренд — плавное долгосрочное изменение уровня ряда. Этую характеристику можно получить, наблюдая ряд в течение достаточно долгого времени.
- Сезонность — циклические изменения уровня ряда с постоянным периодом. В данных о средней зарплате в России (рис. 1.1) очень хорошо видны подобные сезонные колебания: признак всегда принимает максимальное значение в декабре каждого года, а минимальное — в январе следующего года. В целом профиль изменения зарплаты внутри года остаётся более-менее постоянным.
- Цикл — изменение уровня ряда с переменным периодом. Такое поведение часто встречается в рядах, связанных с продажами, и объясняется циклическими изменениями экономической активности. В экономике выделяют циклы длиной 4 – 5 лет, 7 – 11 лет, 45 – 50 лет и т. д. Другой пример ряда с такой характеристикой — это солнечная активность, которая соответствует, например, количеству солнечных пятен за день. Она плавно меняется с периодом, который составляет несколько лет, причём сам период также меняется во времени.
- Ошибка — непрогнозируемая случайная компонента ряда. Сюда включены все те характеристики временного ряда, которые сложно измерить (например, слишком слабые).

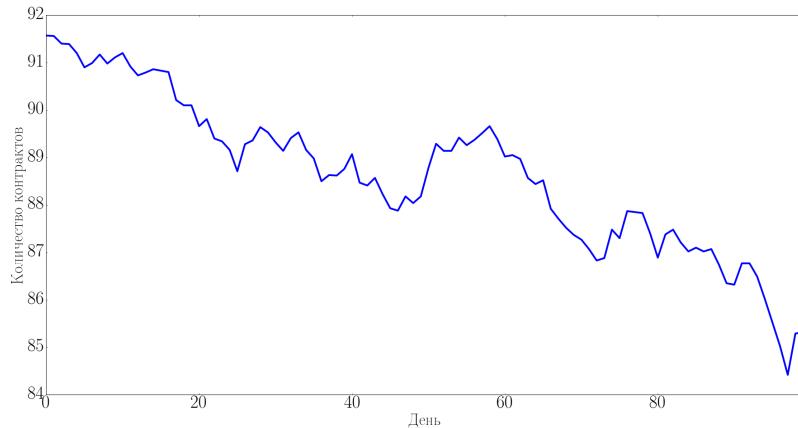


Рис. 1.4: Количество контрактов за день в сокровищнице США

В качестве примера временного ряда можно рассмотреть количество контрактов за день в сокровищнице США (рис. 1.4). На графике виден хорошо выраженный понижающийся тренд, который можно описать линейной функцией. На этом участке в данных не наблюдается ни циклов, ни сезонности. По-видимому, всё, что не удаётся описать трендом, является ошибкой.

На рисунке 1.5 показаны данные за несколько лет о суммарном объёме электричества, произведённого за месяц в Австралии. На графике, как и в предыдущем случае, виден тренд, на этот раз повышающийся. Кроме того, наблюдается годовая сезонность: значение признака совершает колебания, минимум которых всегда приходится на зиму, а максимум — на середину лета. Это легко объяснить тем, что зимой электричества необходимо меньше всего, это самый тёплый сезон в Австралии.

Следующий пример — суммарный объём проданной жилой недвижимости в Америке за месяц (рис. 1.6), данные так же собраны за несколько лет. На графике наблюдается сочетание двух основных компонент. Первая компонента — это годовая сезонность (минимум всегда приходится на зиму, а максимум — на середину лета), а вторая — это циклы, связанные с изменением среднего уровня экономической активности (период в данном случае составляет 7-9 лет).

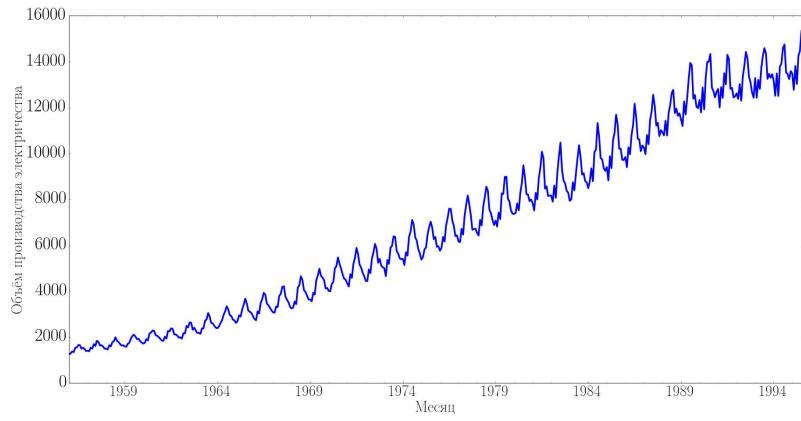


Рис. 1.5: Суммарный объем электричества, произведённого за месяц в Австралии

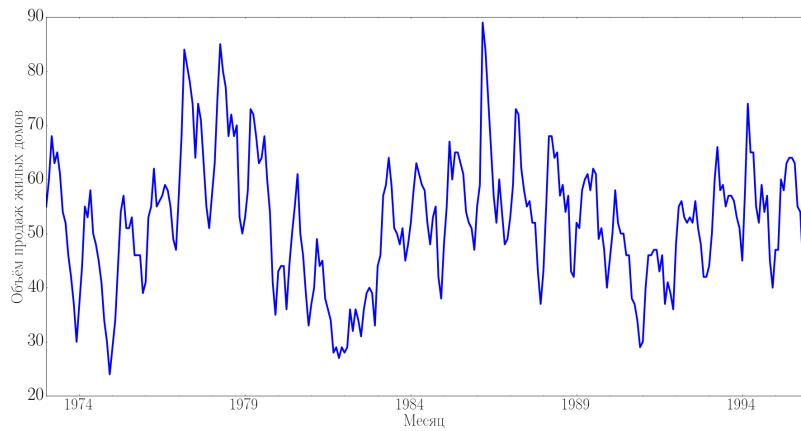


Рис. 1.6: Суммарный объем проданной жилой недвижимости (в млн кв. м.) в Америке за месяц

На рисунке 1.7 показаны ежедневные изменения индекса Доу-Джонса. Глядя на этот график, сложно сказать, присутствует ли в данных какая-то систематическая компонента: явно нет ни тренда, ни сезонности, ни цикла. По всей видимости, ряд представляет собой что-то похожее на случайную ошибку. Однако даже такие ряды можно прогнозировать.

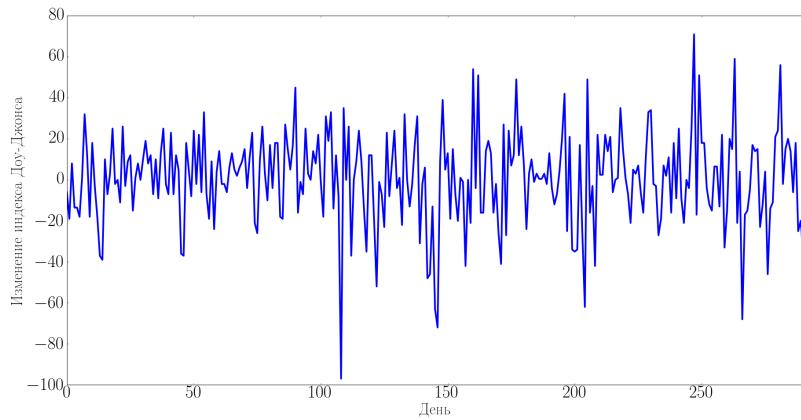


Рис. 1.7: Ежедневное изменение индекса Доу-Джонса

1.2. Автокорреляция

1.2.1. Пример: продажи вина в Австралии

Одной из важнейших характеристик временного ряда является автокорреляция. Далее суть этой характеристики будет демонстрироваться на примере данных о суммарном объёме продаж вина в Австралии за месяц на протяжении почти 15 лет (рис. 1.8).

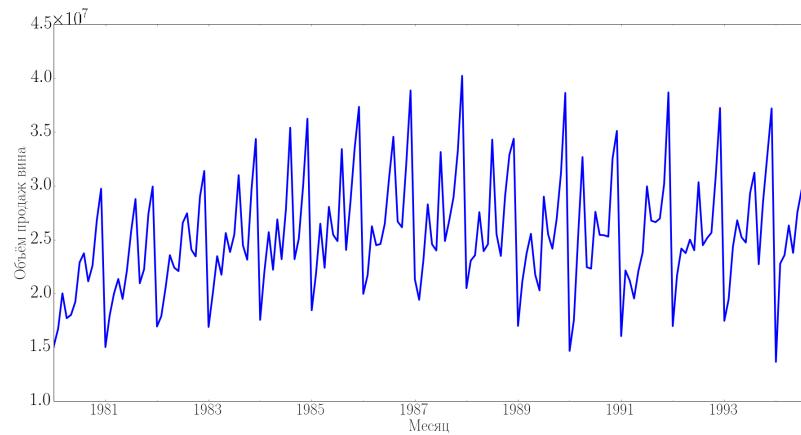


Рис. 1.8: Месячный объём продаж вина в Австралии, в бутылках

Этот ряд обладает ярко выраженной годовой сезонностью: максимум продаж за год приходится на декабрь, а затем, в январе, происходит существенное падение.

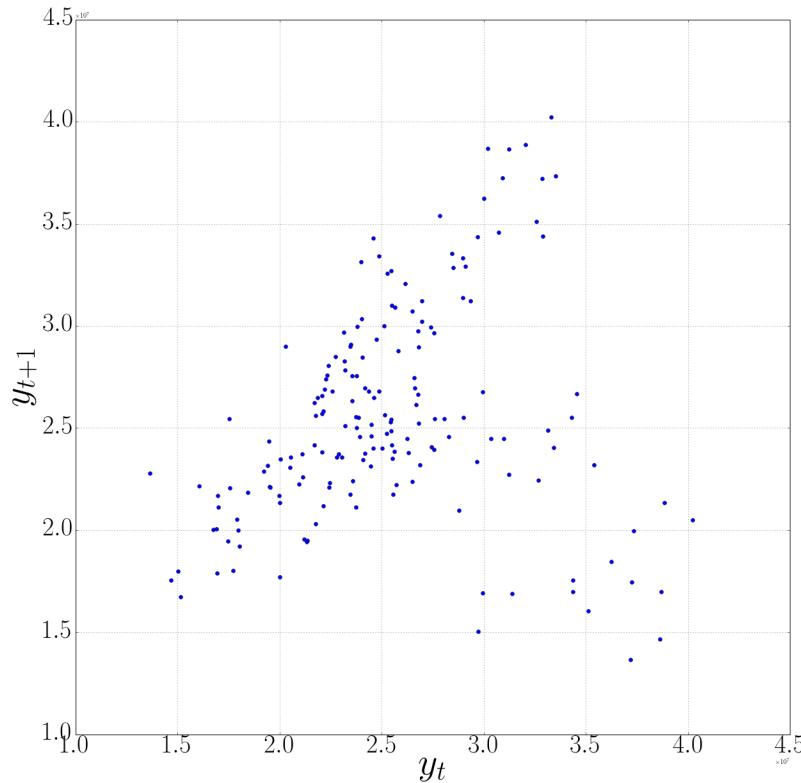


Рис. 1.9: Связь между значениями объёма продаж вина в соседние месяцы, по горизонтали отложен объём продаж в месяц t , по вертикали — в следующий месяц, $t + 1$, каждая точка задаёт продажи в 2 соседних месяца

На рисунке 1.9 показано, как связаны объёмы продаж вина в соседние месяцы. Видно, что большая часть точек на графике группируется вокруг главной диагонали. Это говорит о том, что в основном значения продаж в соседние месяцы похожи. Ещё одно подмножество точек выделяется в правом нижнем углу, оно связано с падением продаж от декабря к январю, которое было видно на предыдущем графике.

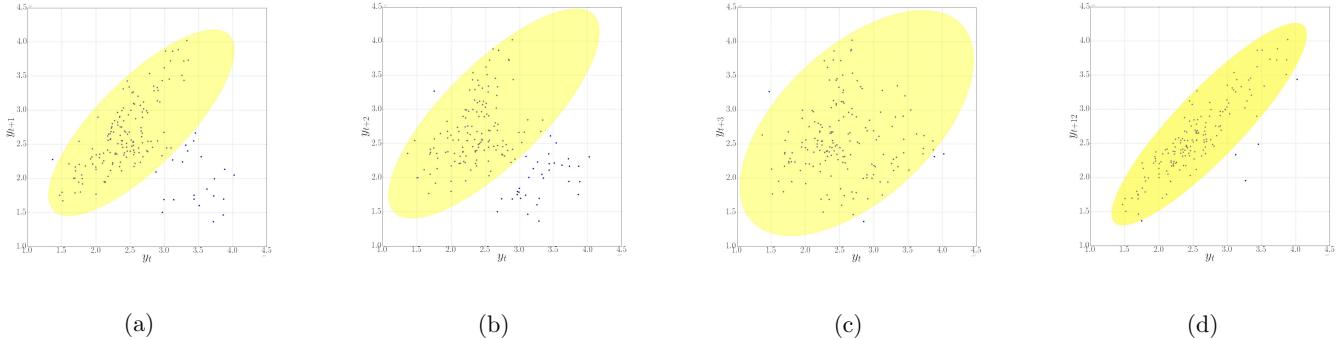


Рис. 1.10: Связь между продажами в соседние месяцы (а), через месяц (б), через два месяца (с) и через год (д).

Если построить аналогичный график, но по вертикальной оси отложить y_{t+2} (рис. 1.10б), то видно, что точки в основном облаке начинают "расплываться" вокруг главной диагонали, то есть сходство между продажами через месяц уменьшается по сравнению с соседними месяцами. Если посмотреть связь между продажами через два месяца (рис. 1.10с), то облако станет ещё шире, а сходство — ещё меньше. Однако если рассмотреть продажи в одни и те же месяцы соседних лет (рис. 1.10д), то видно, что точки на графике снова стягиваются к главной диагонали. Это значит, что значения продаж в одни и те же месяцы соседних лет очень сильно похожи.

1.2.2. Автокорреляция, её вычисление

Количественной характеристикой сходства между значениями ряда в соседних точках является автокорреляционная функция (или просто автокорреляция), которая задаётся следующим соотношением:

$$r_\tau = \frac{\mathbb{E}((y_t - \mathbb{E}y)(y_{t+\tau} - \mathbb{E}y))}{\mathbb{D}y}.$$

Автокорреляция — это уже встречавшаяся ранее корреляция Пирсона между исходным рядом и его версией, сдвинутой на несколько отсчётов. Количество отсчётов, на которое сдвинут ряд, называется лагом автокорреляции (τ). Значения, принимаемые автокорреляцией такие же, как и у коэффициента Пирсона: $r_\tau \in [-1, 1]$.

Вычислить автокорреляцию по выборке можно, заменив в формуле математическое ожидание на выборочное среднее, а дисперсию — на выборочную дисперсию:

$$r_\tau = \frac{\sum_{t=1}^{T-\tau} (y_t - \bar{y})(y_{t+\tau} - \mathbb{E}y)}{\sum_{t=1}^{T-\tau} ((y_t - \bar{y}))^2}.$$

1.2.3. Коррелограммы

Анализировать величину автокорреляции при разных значениях лагов удобно с помощью графика, который называется коррелограммой. По оси ординат на нём откладывается автокорреляция, а по оси абсцисс — размер лага τ . На рисунке 1.11а показан пример коррелограммы для исследуемых ранее данных о месячных продажах вина в Австралии (рисунок 1.8). На графике видно, что автокорреляция принимает большие значения в лагах, кратных сезонному периоду. Такой вид коррелограммы типичен для данных с выраженной сезонностью.

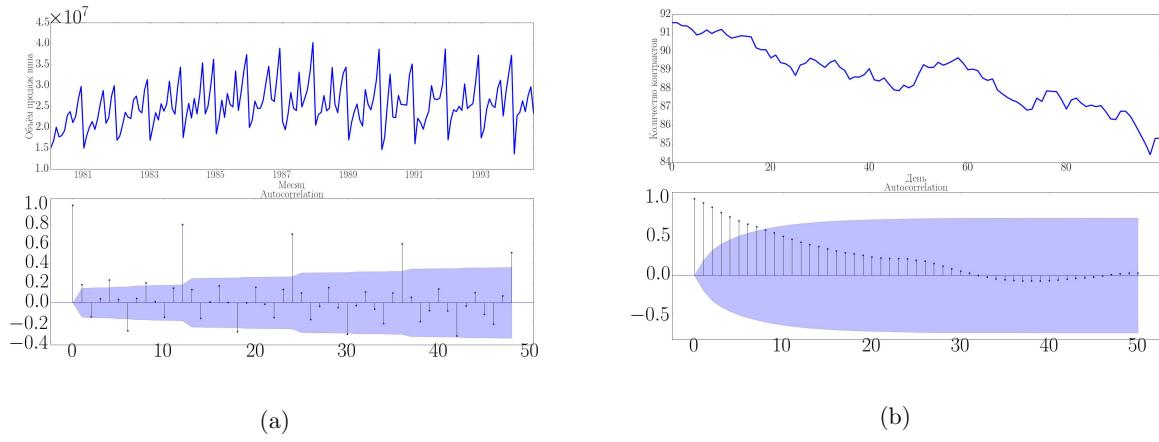


Рис. 1.11: Коррелограммы для временных рядов: (а) — количество проданного вина в Австралии за месяц, (б) — количество контрактов за день в сокровищнице США

На рисунке 1.11b показано, как выглядит коррелограмма для данных с ярко выраженным трендом. Автокорреляция тем больше, чем меньше величина лага τ , и с ростом τ она начинает постепенно убывать, при этом автокорреляция может начать колебаться вокруг горизонтальной оси, соответствующей её нулевому значению.

Коррелограмма, изображённая на рисунке 1.12a, построены для временного ряда, в котором присутствуют и тренд, и сезонность. Таким образом, на ней можно наблюдать оба описанных ранее эффекта, однако тренд настолько сильный, что практически нейтрализует влияние сезонности (следствие которой — наличие пиков в лагах, кратных периоду сезона).

На рисунке 1.12b показана типичная коррелограмма для ряда, в котором есть и сезонность, и цикл. Для самого первого лага, кратного сезонному периоду, виден пик, однако далее положение этого пика смещается: следующий пик не приходится на 2, 3 или 4 года. Это происходит, потому что в ряде есть циклы, период которых плавно меняется.

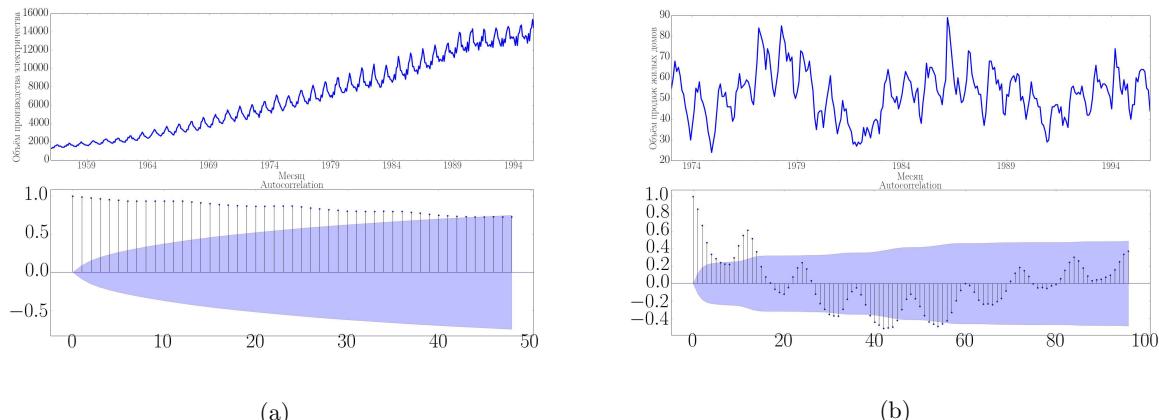


Рис. 1.12: Коррелограммы для временных рядов: (а) — ежемесячное производство электричества в Австралии, (б) — объём проданной в Америке недвижимости за месяц

На коррелограмме, соответствующей данным о ежедневном изменении индекса Доу-Джонса, все значения автокорреляции невелики, кроме первого (в данной точке лаг $\tau = 0$, и вычисляется корреляция значения ряда с самим собой, а такая корреляция всегда равна 1).

1.2.4. Значимость автокорреляции

На всех показанных коррелограммах помимо значений автокорреляции также изображен синий коридор вокруг горизонтальной оси. Это коридор значимости отличия корреляции от нуля. Как правило, его выводят

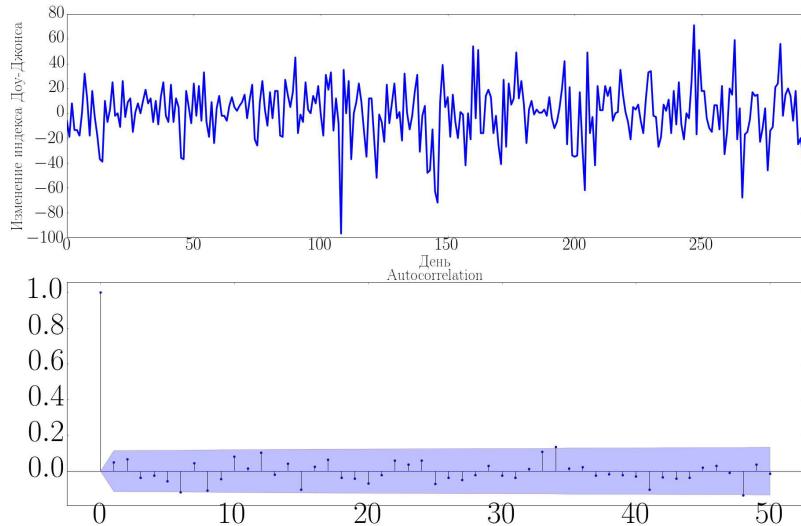


Рис. 1.13: Коррелограмма для данных о ежедневном изменении индекса Доу-Джонса.

на график все стандартные средства для работы с временными рядами. Фактически, все автокорреляции, которые изображены вне этого коридора, значимо отличаются от нуля. Как и для обычной корреляции Пирсона, значимость вычисляется с помощью критерия Стьюдента (таблица 1.1). Альтернатива чаще всего двусторонняя, потому что при анализе временных рядов крайне редко имеется гипотеза о том, какой должна быть корреляция, положительной или отрицательной.

временной ряд:	$y^T = y_1, \dots, y_T$
нулевая гипотеза:	$H_0: r_\tau = 0$
альтернатива:	$H_1: r_\tau < \neq > 0$
статистика:	$T(y^T) = \frac{r_\tau \sqrt{T-\tau-2}}{\sqrt{1-r_\tau^2}}$
нулевое распределение:	$T(y^T) \sim St(T-\tau-2)$.

Таблица 1.1: Описание статистического критерия Стьюдента

Вернувшись к коррелограмме по данным о ежедневном изменении индекса Доу-Джонса, теперь можно заметить, что ни одна из корреляций не выходит за пределы коридора значимости, а значит ни одна из них не является значимо отличающейся от нуля.

1.3. Стационарность

1.3.1. Понятие стационарности временного ряда

Ещё одно важное свойство временных рядов — это стационарность. Временной ряд y_1, \dots, y_T называется стационарным, если $\forall s$ (ширина окна) распределение y_t, \dots, y_{t+s} не зависит от t , т.е. его свойства не зависят от времени.

Из этого определения следует, что ряды, в которых присутствует тренд, являются нестационарными: в зависимости от расположения окна изменяется средний уровень ряда. Кроме того, нестационарны ряды с сезонностью: если ширина окна меньше сезонного периода, то распределение ряда будет разным, в зависимости от положения окна. При этом интересно, что ряды, в которых есть непериодические циклы, не обязательно являются нестационарными, поскольку нельзя заранее предсказать положение максимумов и минимумов этого ряда.

В качестве примера стационарных и нестационарных можно рассмотреть временные ряды, показанные на рисунке 1.14. Ряды 1.14a, 1.14c, 1.14e, 1.14f, 1.14i не являются стационарными из-за довольно выраженного тренда. В рядах 1.14d, 1.14h, 1.14i сильно выражена сезонность, поэтому они также не стационарны. В ряду 1.14i, помимо всего прочего, меняется и дисперсия (размах колебаний в начале ряда намного меньше, чем в конце), то есть присутствует ещё одно свойство, не постоянное по времени. Остаются два ряда, 1.14b и 1.14g.

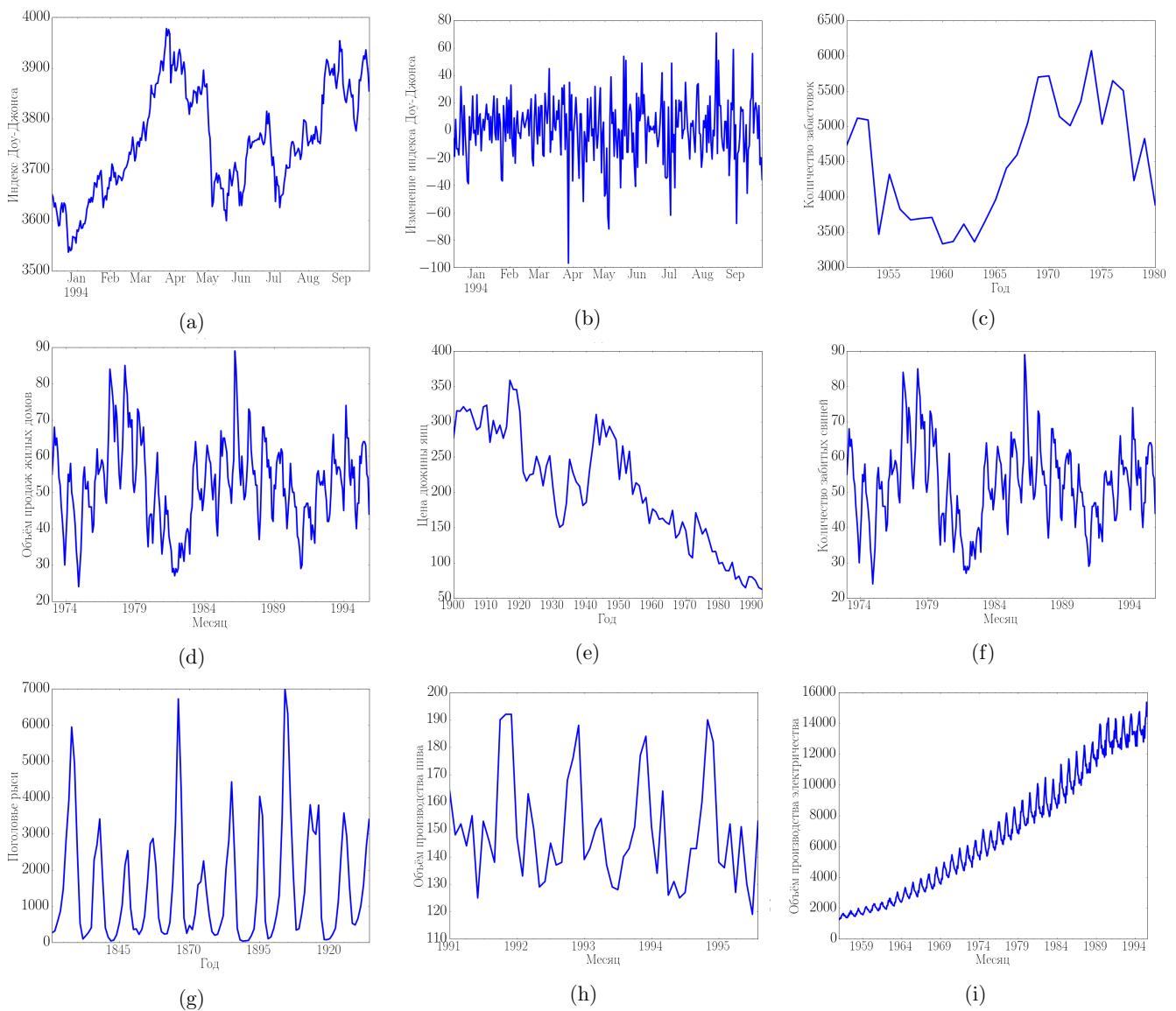


Рис. 1.14: Примеры временных рядов

Первый — это ежедневное изменение индекса Доу-Джонса, этот ряд считается стационарным, а второй — это размер поголовья рыси. Колебания во втором ряду имеют нефиксированный период, то есть это ряд с циклами, значит, он может считаться стационарным.

1.3.2. Критерий Дики-Фуллера

Формально гипотезу о стационарности можно проверить с помощью критерия Дики-Фуллера (таблица 1.2). Статистика данного критерия выглядит достаточно сложно и пока что рассматриваться не будет.

Вообще говоря, существует большое количество критериев для проверки гипотезы о стационарности, на практике можно использовать любой из них. Однако в этом курсе внимание акцентируется на критерии Дики-Фуллера, потому что для него существует реализация в языке Python.

1.3.3. Стабилизация дисперсии

При работе с нестационарными временными рядами используется ряд стандартных трюков, чтобы сделать их стационарными. В случае, если во временном ряде монотонно по времени изменяется дисперсия, применяется специальное преобразование, стабилизирующее дисперсию. Очень часто в качестве такого преобразования

временной ряд:	$y^T = y_1, \dots, y_T$
нулевая гипотеза:	H_0 : ряд нестационарен
альтернатива:	H_1 : ряд стационарен
статистика:	неважно
нулевое распределение:	табличное

Таблица 1.2: Описание статистического критерия Дики-Фуллера

ния выступает логарифмирование. Результат стабилизации дисперсии для ряда производства электричества в Австралии показан на рисунке 1.15. Видно, что после логарифмирования размах колебаний в начале и конце ряда становится очень похожим, и дисперсия примерно стабилизируется.

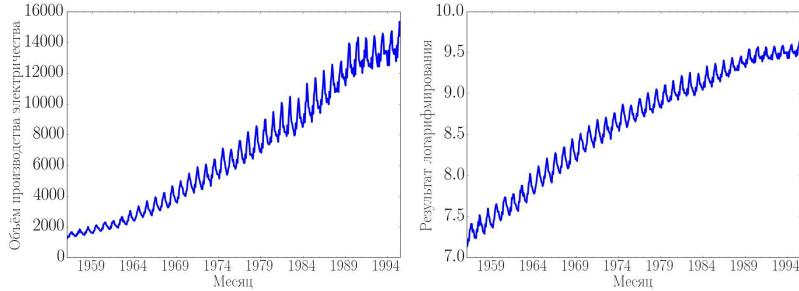


Рис. 1.15: Данные о производстве электричества в Австралии до и после логарифмирования.

Логарифмирование принадлежит к семейству преобразований Бокса-Кокса.

$$y'_t = \begin{cases} \ln y_t, & \lambda = 0, \\ (y_t^\lambda - 1) / \lambda, & \lambda \neq 0. \end{cases}$$

Это параметрическое семейство функций, в котором параметр λ определяет, как именно будет преобразован ряд: $\lambda = 0$ — это логарифмирование, $\lambda = 1$ — тождественное преобразование ряда, а при других значениях λ — степенное преобразование. Значение параметра можно подбирать так, чтобы дисперсия была как можно более стабильной во времени. Так, для ряда по данным производства электричества в Австралии оптимальное значение $\lambda = 0.27$, при этом дисперсия немного более стабильна, чем при логарифмировании (рис. 1.16).

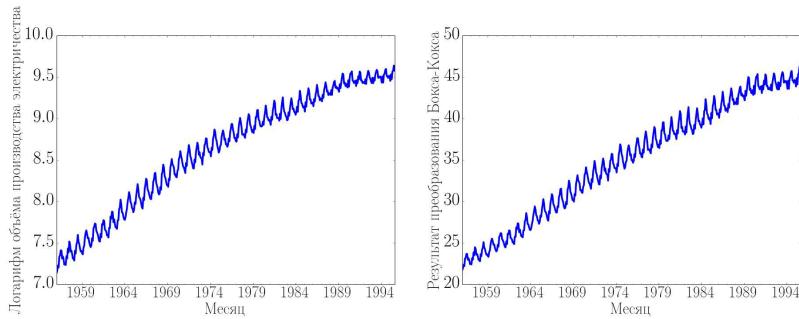


Рис. 1.16: Преобразованный временной ряд по данным о производстве электричества в Австралии. Слева — результат логарифмирования, справа — преобразование Бокса-Кокса с параметром $\lambda = 0.27$.

1.3.4. Дифференцирование

Ещё один важный трюк, который позволяет сделать ряд стационарным, — это дифференцирование, переход к попарным разностям соседних значений:

$$y' = y_t - y_{t-1}.$$

Для нестационарного ряда часто оказывается, что получаемый после дифференцирования ряд является стационарным. Такая операция позволяет стабилизировать среднее значение ряда и избавиться от тренда, а иногда даже от сезонности. Кроме того, дифференцирование можно применять неоднократно: от ряда первых разностей, продифференцировав его, можно прийти к ряду вторых разностей, и т. д. Длина ряда при этом каждый раз будет немного сокращаться, но при этом он будет стационарным.

Также может применяться сезонное дифференцирование ряда, переход к попарным разностям значений в соседних сезонах. Если длина периода сезона составляет s , то новый ряд задаётся разностями

$$y'_t = y_t - y_{t-s}.$$

Сезонное и обычное дифференцирование могут применяться к ряду в любом порядке. Однако если у ряда есть ярко выраженный сезонный профиль, то рекомендуется начинать с сезонного дифференцирования, уже после такого преобразования может оказаться, что ряд стационарен.

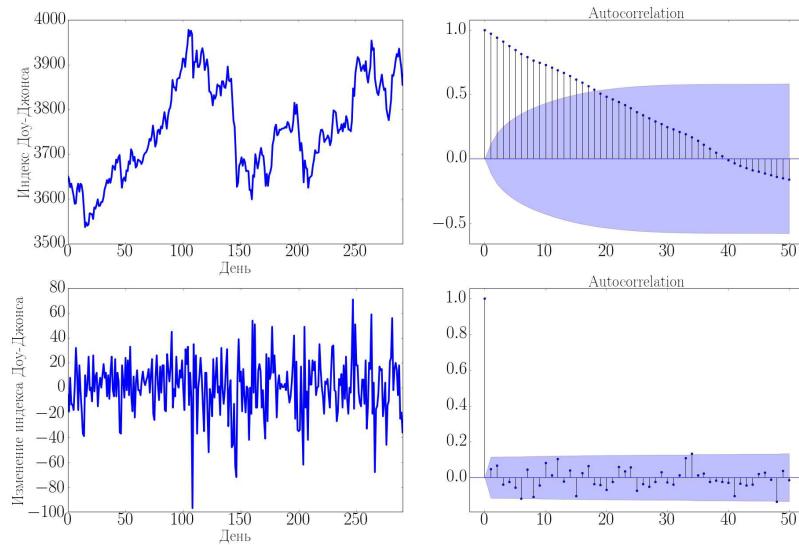


Рис. 1.17: Данные о значении индекса Доу-Джонса и соответствующая коррелограмма. Сверху — исходный ряд, снизу — ряд после дифференцирования.

На верхних графиках на рисунке 1.17 показаны ряд значений индекса Доу-Джонса и его автокорреляционная функция. Видно, что этот ряд достаточно сильно нестационарен — имеется ярко выраженный тренд. От этого тренда удается полностью избавиться, продифференцировав ряд. Снизу на рисунке 1.17 показан ряд после дифференцирования (это ежедневное изменение индекса, и этот ряд уже встречался ранее). Критерий Дики-Фуллера подтверждает, что новый ряд, полученный дифференцированием, является стационарным. Нулевая гипотеза о нестационарности этого ряда отвергается ($p = 5.2 \times 10^{-29}$). Для исходного ряда отвергнуть нулевую гипотезу не удается ($p = 0.3636$).

1.4. ARMA

Первый класс прогнозирующий моделей, который будет разбираться в этом курсе, — это модели ARMA.

1.4.1. Авторегрессия

Ранее была предпринята попытка свести задачу прогнозирования временного ряда к задаче обучения с учителем: предсказывать значения ряда с помощью регрессии, выбирая какие-то признаки, зависящие от времени (например, линейный или квадратичный тренд, рис. 1.2). Результат получился плохим, выбранных признаков явно недостаточно, нужны дополнительные.

Можно перейти к следующей идеи: делать регрессию для ряда не на какие-то внешние признаки, зависящие от времени, а на его собственные значения в прошлом:

$$y_t = \alpha + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t.$$

В этом регрессионном уравнении y_t — это отклик, $y_{t-1}, y_{t-2}, \dots, y_{t-p}$ — признаки, $\alpha, \phi_1, \phi_2, \dots, \phi_p$ — параметры модели, которые необходимо оценить, ε_t — шумовая компонента, описывает отклонения значений ряда от данного уравнения.

Такая модель называется моделью авторегрессии порядка p ($AR(p)$). В этой модели y_t представляет собой линейную комбинацию p предыдущих значений ряда и шумовой компоненты.

1.4.2. Скользящее среднее

Следующий класс моделей — это скользящее среднее. Чтобы лучше понимать, как они устроены, можно начать с рассмотрения независимого, одинаково распределённого во времени шума ε_t (рис. 1.18a).

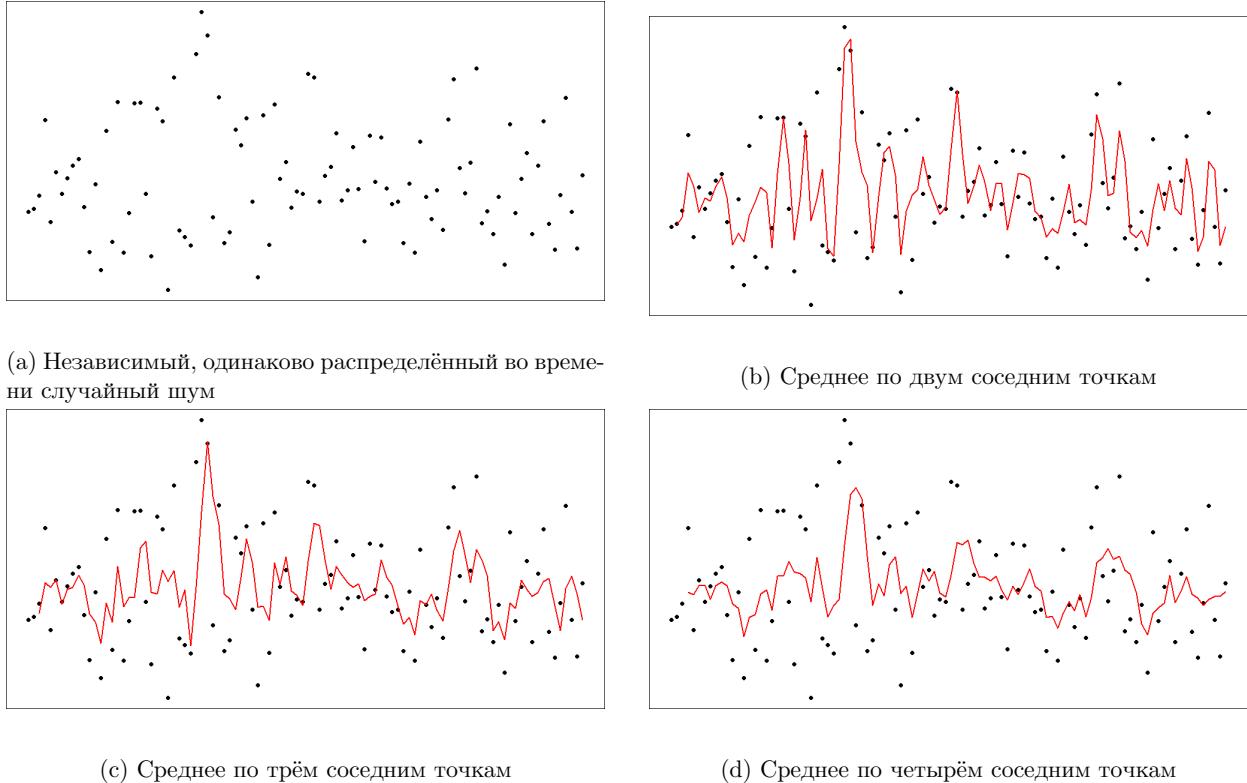


Рис. 1.18

Для каждого значения t можно вычислить среднее арифметическое между точками ε_t и ε_{t-1} (рис. 1.18b). Также можно вычислять среднее не по двум, а по трём (рис. 1.18c) или четырём (рис. 1.18d) точкам. То, что получается в результате такого усреднения, — это уже не простая выборка с независимыми, одинаково распределёнными элементами. Соседние значения на красной линии очень похожи друг на друга, потому что в их вычислении используются одни и те же шумовые компоненты.

Данную идею можно обобщить и записать следующую модель ряда:

$$y_t = \alpha + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q},$$

где $\varepsilon_t, \varepsilon_{t-1}, \dots, \varepsilon_{t-q}$ — значения шума в q предыдущих моментов времени, $\alpha, \theta_1, \theta_2, \dots, \theta_q$ — это параметры модели, которые необходимо оценить. Такая модель называется моделью скользящего среднего порядка q ($MA(q)$). В ней предполагается, что значение ряда y_t — это линейная комбинация q последних значений шумовой компоненты.

Данная модель выглядит достаточно странно: шумовая компонента — это что-то, что невозможно наблюдать, и непонятно, как эту модель обучать, и зачем она нужна. Ответы на эти вопросы будут даны далее.

1.4.3. ARMA

Можно проделать следующий трюк: взять авторегрессионную модель порядка p ($AR(p)$) и модель скользящего среднего порядка q ($MA(q)$) и сложить то, что находится у них в правых частях. Результат — это

модель $ARMA(p, q)$, она выглядит следующим образом:

$$y_t = \alpha + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}.$$

Главное, что нужно знать об этой модели: теорема Вольда утверждает, что любой стационарный временной ряд может быть описать моделью $ARMA(p, q)$ с правильным подбором значений параметров p, q . Это прекрасный результат, который означает, что семейство моделей $ARMA(p, q)$ достаточно богато для того, чтобы в нём можно было найти хорошую модель, описывающую любой стационарный ряд.

1.4.4. Пример

Для демонстрации работы модели $ARMA(p, q)$ можно рассмотреть данные о поголовье рыси (рисунок 1.19). Этот пример уже встречался ранее, и было показано, что ряд стационарен, а значит в классе $ARMA(p, q)$ для него можно найти достаточно хорошее описание.

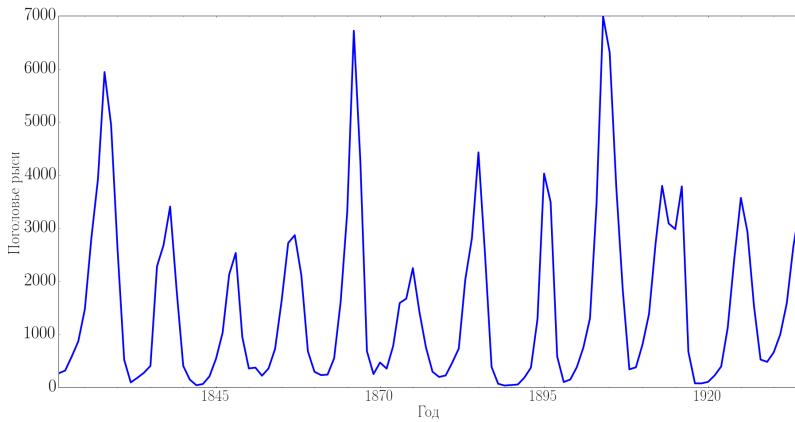


Рис. 1.19: Данные о размере поголовья рыси.

Действительно, модель $ARMA(2, 2)$ даёт результат, который достаточно сильно похож на исходный ряд. Модель не во всех точках близка к истинному значению ряда, однако результат всё равно намного лучше, чем если бы для приближения использовалась регрессия на линейный или квадратичный временной тренд.

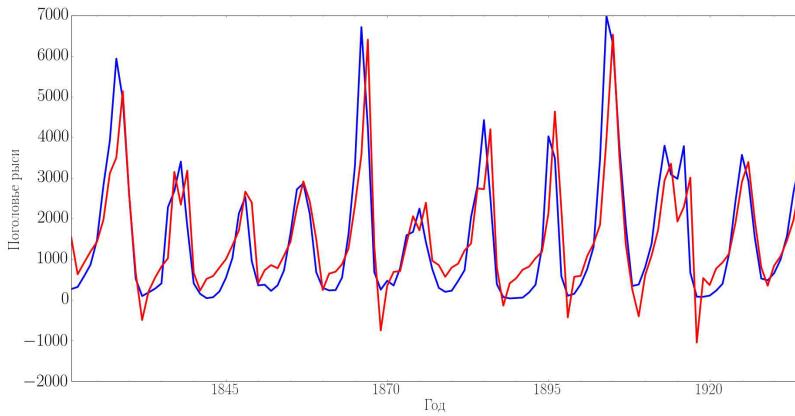


Рис. 1.20: Результат применения модели $ARMA(2, 2)$ к данным о поголовье рыси.

Настроив модель $ARMA(2, 2)$, её можно использовать и для построения прогноза (рис. 1.21), то есть решения той задачи, которая была изначально поставлена. Однако вопрос о том, какую модель использовать для прогнозирования нестационарного временного ряда, остаётся открытым.

1.5. ARIMA

Модели типа ARIMA — это обобщение модели класса ARMA.

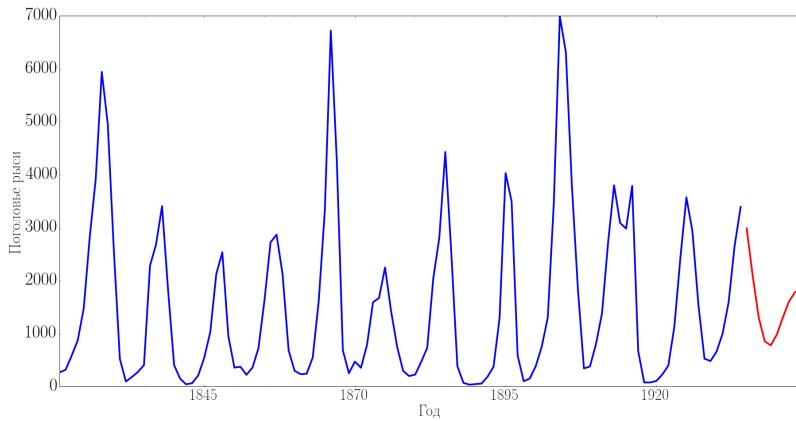


Рис. 1.21: Результат применения модели $ARMA(2, 2)$ для получения прогноза

На данный момент известны два факта:

1. Теорема Вольда: любой стационарный ряд может быть описан моделью $ARMA(p, q)$ с любой наперёд заданной точностью.
2. При помощи дифференцирования нестационарный ряд можно сделать стационарным.

Эти две идеи и лежат в основе моделей класса ARIMA. Модель $ARIMA(p, d, q)$ — это модель $ARMA(p, q)$ для d раз продифференцированного ряда.

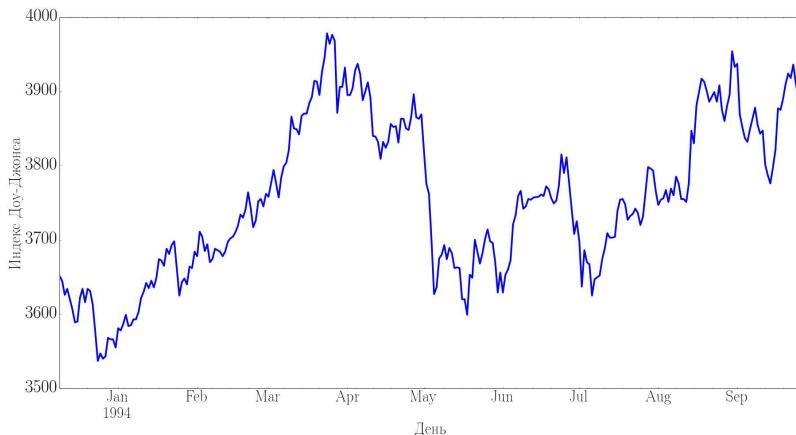


Рис. 1.22: Данные о значении индекса Доу-Джонса

На рисунке 1.22 показаны 300 значений индекса Доу-Джонса. Ранее было показано, что ряд нестационарен (и это видно невооружённым глазом), но зато стационарен ряд его первых разностей. Из этого следует, что для ряда разностей можно подобрать достаточно хорошую модель в классе ARMA. Если сделать это, а затем произвести операцию, обратную дифференцированию, то в результате будет получена модель ARIMA для исходного ряда.

Модель $ARIMA(0, 1, 0)$ для значения индекса Доу-Джонса показана на рисунке 1.23. В этой модели происходит одно дифференцирование и не используется ни одной компоненты авторегрессии и скользящего среднего, и это немного странно. Ряд разностей моделируется константой, но после проведения операции, обратной дифференцированию, полученный результат не является константой. В этой модели много странностей, но результат в любом случае лучше, чем то, что можно было бы получить с помощью регрессии ряда на временные признаки.

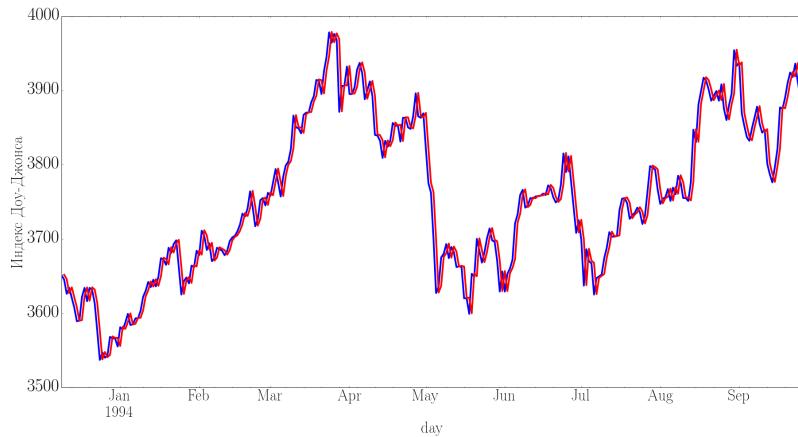


Рис. 1.23: Модель $ARIMA(0, 1, 0)$, применённая к ряду значений индекса Доу-Джонса.

1.5.1. SARMA

Настало время разобраться с сезонностью. Пусть ряд имеет сезонный период длины S . Тогда можно взять модель $ARMA(p, q)$:

$$y_t = \alpha + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

добавить к этой модели P авторегрессионных компонент, но не предыдущих, а с шагом, равным периодом сезонности:

$$+\phi_S y_{t-S} + \phi_{2S} y_{t-2S} + \cdots + \phi_{PS} y_{t-PS}$$

и Q компонент скользящего среднего, также с шагом, равным периодом сезонности:

$$+\theta_S \varepsilon_{t-S} + \theta_{2S} \varepsilon_{t-2S} + \cdots + \theta_{PS} \varepsilon_{t-QS}.$$

Результат — это модель $SARMA(p, q) \times (P, Q)$.

1.5.2. SARIMA

Модель $SARIMA(p, d, q) \times (P, D, Q)$ — модель $SARMA(p, q) \times (P, Q)$ для ряда, к которому d раз было применено обычное дифференцирование и D раз — сезонное. Такую модель часто называют просто ARIMA: первая буква не пишется, но подразумевается, что сезонная компонента тоже может быть (такая терминология будет встречаться и дальше в курсе).

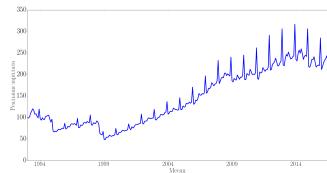
1.5.3. Пример

Для демонстрации рассмотренных моделей будет использоваться временной ряд реальной заработной платы в России (рис. 1.24a). Критерий Дики-Фуллера не отвергает гипотезу о том, что этот ряд нестационарный ($p = 0.2265$). Это неудивительно, потому что видно, что многие параметры этого ряда меняются во времени. Во-первых, меняется дисперсия: разброс скачков в начале совсем не такой, как ближе к концу.

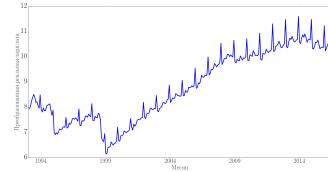
Ряд после применения преобразования Бокса-Кокса с параметром $\lambda = 0.22$ показан на рисунке 1.24b. Критерий Дики-Фуллера всё ещё не отвергает для этого ряда гипотезу о нестационарности ($p = 0.1661$). Это можно объяснить наличием в ряду сезонности и тренда.

После применения к ряду сезонного дифференцирования (рис. 1.24c) критерий Дики-Фуллера отвергает гипотезу о нестационарности ($p = 0.01$). Относительно этого ряда можно говорить, что он стационарный, а значит, можно попытаться подобрать для него модель в классе ARMA или даже сезонную модель. Если после этого провести обратные преобразования к преобразованию Бокса-Кокса и сезонному дифференцированию, модель может выглядеть, например, как на рисунке 1.24d. Красная линия на графике — это предсказание модели, видно, что она достаточно хорошо описывает исходные данные, а значит, можно надеяться, что и прогнозы она будет давать хорошие.

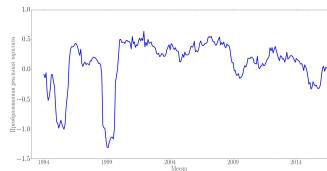
При применении регрессии с линейным или квадратичным трендом по времени в остатках этой модели было видно достаточно много структуры (рис. 1.3), а значит, что в данных оставалось много информации, которую не учитывает модель.



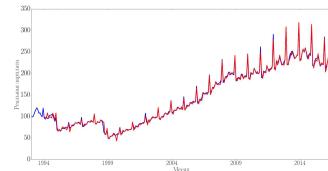
(a) Реальная месячная заработная плата в России.



(b) Ряд реальной месячной заработной платы в России после применения преобразования Бокса-Кокса.



(c) Ряд реальной месячной заработной платы в России после применения сезонного дифференцирования.



(d) Синим — ряд реальной месячной заработной платы, красным — модель SARIMA(2, 0, 2) \times (0, 1, 0) с преобразованием Бокса-Кокса.

Рис. 1.24

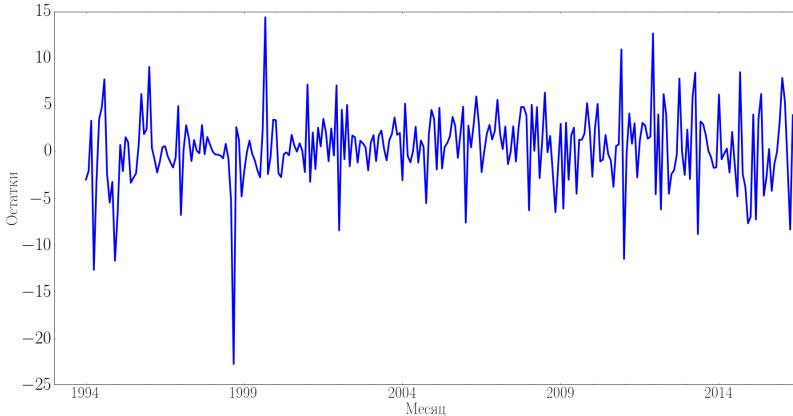


Рис. 1.25: Остатки построенной модели SARIMA.

Остатки для построенной модели SARIMA показаны на рисунке 1.25. Они уже гораздо больше похожи на белый шум. Выброс в остатках — это кризис 1998 года, который плохо описывается построенной моделью. Тем не менее, в этих остатках уже практически не имеется структуры, а значит, полученный результат лучше, чем при использовании линейной регрессии.

1.6. Выбор ARIMA и прогнозирование

После изучения устройства моделей класса ARIMA настало время разобраться, как настраивать эти модели и получать с их помощью прогнозы.

1.6.1. Подбор параметров

У моделей класса ARIMA есть несколько групп параметров. Параметры d, D, q, Q, p, P можно считать гиперпараметрами, поскольку они определяют структуру и количество коэффициентов в самой модели ARIMA. Остальные параметры, α, ϕ, θ , являются коэффициентами в регрессионном уравнении.

Параметры α, ϕ, θ

Если зафиксированы параметры d, D, q, Q, p, P , то есть зафиксирована структура модели ARIMA, то параметры α, ϕ, θ можно подобрать с помощью метода наименьших квадратов. Фактически происходит на-

страивание привычной регрессии методом минимизации квадратичной ошибки.

Единственный трюк заключается в определении коэффициентов θ , которые стоят при шумовых компонентах из прошлого. Наблюдать шумовые компоненты невозможно, поэтому, чтобы подставить их в регрессионное уравнение, их нужно предварительно оценить. Обычно оценка производится с помощью остатков от авторегрессии, которая предварительно строится по исследуемым данным.

Если шум, который стоит в модели ARIMA, является белым (независимый, одинаково распределённый, гауссовский), то метод наименьших квадратов даёт оценки максимального правдоподобия для параметров α, ϕ, θ , то есть заведомо известно, что эти оценки хороши.

Параметры d, D

Параметры d, D , которые задают порядки дифференцирования, необходимо подбирать так, чтобы ряд стал стационарным. Ранее уже упоминалось, что всегда рекомендуется начинать с сезонного дифференцирования, потому что уже после него ряд может оказаться стационарным. Дело в том, что выгодно дифференцировать ряд как можно меньше раз, потому что с увеличением количества дифференцирований растёт дисперсия итогового прогноза.

Параметры q, Q, p, P

К сожалению, гиперпараметры q, Q, p, P нельзя выбирать из принципа максимума правдоподобия. Например, чем больше значение параметра p , тем больше авторегрессионных компонент в итоговом уравнении, тем больше параметров ϕ и тем лучше это уравнение описывает данные. Чем больше значения гиперпараметров, тем больше параметров в модели и тем она сложнее. Таким образом, с увеличением значения этих гиперпараметров значение правдоподобия может только увеличиваться. Поэтому для сравнения моделей с разным количеством параметров необходим другой критерий.

В качестве искомого критерия можно использовать, например, критерий Акаике:

$$AIC = -2 \ln L + 2k,$$

где L — правдоподобие, $k = P + Q + p + q + 1$ — число параметров в модели.

Оптимальной по критерию Акаике будет модель с наименьшим значением этого критерия. Такая модель, с одной стороны, будет достаточно хорошо описывать данные, а с другой — содержать не слишком большое количество параметров.

В конечном итоге значения параметров q, Q, p, P определяются перебором: из разных значений гиперпараметров выбираются те, у которых значение критерия Акаике будет минимальным. Начальные приближения для этого перебора можно выбрать с помощью автокорреляционной функции.

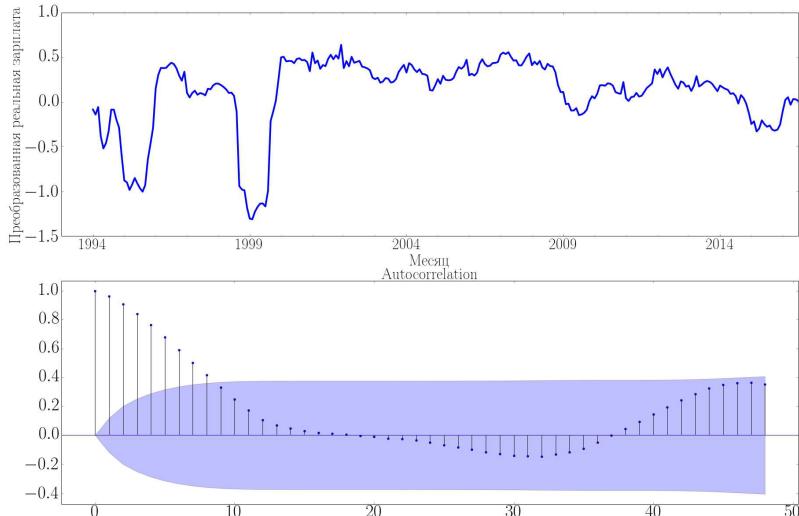


Рис. 1.26: Сверху — ряд реальной заработной платы в России после преобразования Бокса-Кокса и сезонного дифференцирования, снизу — автокорреляционная функция этого ряда.

Принцип подбора параметров можно продемонстрировать на данных о реальной заработной плате в России (1.26). Начальное значение для параметра $Q * S$ даёт номер последнего сезонного лага, при котором автокорреляция значима. В рассматриваемом примере сезонных лагов со значимой корреляцией нет, значит, начальное приближение $Q = 0$. Параметр q задаётся номером последнего несезонного лага, при котором автокорреляция значима. В данном случае можно взять начальное значение $q = 8$.

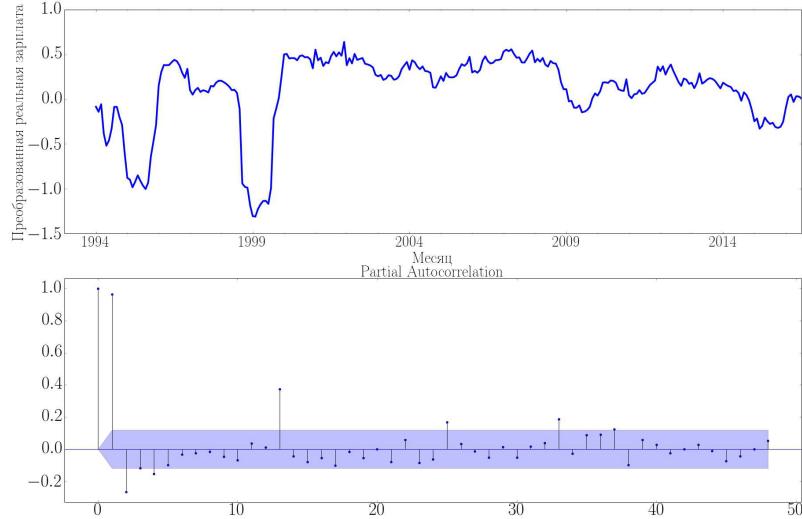


Рис. 1.27: Сверху — ряд реальной заработной платы в России после преобразования Бокса-Кокса и сезонного дифференцирования, снизу — частичная автокорреляционная функция этого ряда.

Значения параметров p, P подбираются с использованием не автокорреляционной функции, а частичной автокорреляционной функции (рис. 1.27). Частичная автокорреляция — это автокорреляция после снятия авторегрессии предыдущего порядка. Например, чтобы подсчитать частичную автокорреляцию с лагом $\tau = 2$, требуется построить авторегрессию порядка 1, вычесть эту авторегрессию из ряда и подсчитать автокорреляцию на полученных остатках.

Начальное приближение для параметра $P * S$ задаёт номер последнего сезонного лага, при котором частичная автокорреляция значима. В данных из примера это лаг под номером 24, значит начальное приближение $P = 2$, поскольку длина сезонного периода $S = 12$. Аналогично, p задаётся как номер последнего несезонного лага, при котором частичная автокорреляция значима. В данном случае можно взять начальное приближение $p = 2$.

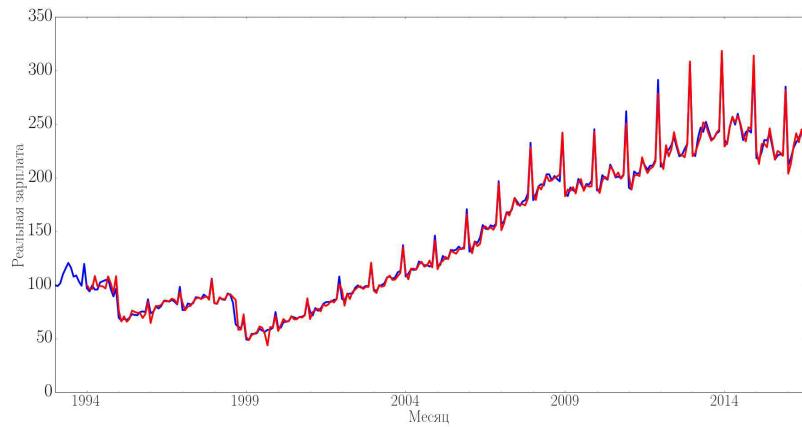


Рис. 1.28: Модель $ARIMA(2, 0, 1) \times (2, 1, 2)$, которая по критерию Акаике оптимально описывает данные о реальной заработной плате в России

Обобщая всю вышеизложенную информацию о прогнозировании ряда реальной заработной платы в России, далее необходимо перебрать разные модели в классе $ARIMA$ со значениями параметров $D = 1, d = 0$

и преобразованием Бокса-Кокса, начиная с начальных приближений, которые были получены из автокорреляционной функции. Сравнение моделей будет производиться по информационному критерию Акаике. В результате, самая лучшая по этому критерию модель, — это $ARIMA(2, 0, 1) \times (2, 1, 2)$ (рис. 1.28).

1.6.2. Подбор ARIMA

Итоговый алгоритм подбора модели в классе ARIMA состоит в следующем:

- В первую очередь необходимо построить график ряда и посмотреть на него. Уже из визуального анализа можно сделать определённые выводы: есть ли в данных сезонность, какой сезонный период, есть ли в ряде пропуски и выбросы, необходимо ли стабилизировать дисперсию, стоит ли исключить из рассмотрения начало ряда, потому что значения в начале совсем не похожи на значения в конце.
- Следующий шаг — это стабилизация дисперсии при необходимости. Стабилизация производится с помощью метода Бокса-Кокса или логарифмированием, что является частным случаем того же метода.
- Если исследуемый ряд нестационарен, необходимо подобрать порядок дифференцирования, при котором он становится стационарным. Таким образом фиксируются параметры d, D модели ARIMA.
- Далее необходимо построить графики автокорреляционной функции (ACF) и частичной автокорреляционной функции (PACF) и из этих графиков определить примерные значения параметров p, q, P, Q . Фактически эти значения — начальные приближения, с которых начинается перебор разных моделей.
- Полученные модели необходимо обучить, сравнить их по информационному критерию Акаике и выбрать ту, которая его минимизирует.
- Необходимо посмотреть на остатки получившейся модели, чтобы понять, насколько хорошей она получилась, можно ли, теоретически, её улучшить, нет ли в ней каких-то видимых недостатков. Подробнее об анализе остатков будет рассказано позднее.

1.6.3. Прогнозирование

Теперь необходимо разобраться, как на основании настроенной модели ARIMA правильно строить прогноз. Пусть модель построена, определены значения всех неизвестных параметров, получены их оценки $\hat{\alpha}, \hat{\phi}, \hat{\theta}$, которые записаны в этом уравнении:

$$y_t = \hat{\alpha} + \hat{\phi}_1 y_{t-1} + \cdots + \hat{\phi}_p y_{t-p} + \varepsilon_t + \hat{\theta}_1 \varepsilon_{t-1} + \cdots + \hat{\theta}_q \varepsilon_{t-q}.$$

Чтобы построить прогноз на момент времени $T + 1$, нужно в этом уравнении заменить все индексы t на $T + 1$:

$$\hat{y}_{T+1|T} = \hat{\alpha} + \hat{\phi}_1 y_T + \cdots + \hat{\phi}_p y_{T+1-p} + \varepsilon_{T+1} + \hat{\theta}_1 \varepsilon_T + \cdots + \hat{\theta}_q \varepsilon_{T+1-q}.$$

В этом уравнении присутствует значение ошибки из будущего ε_{T+1} . Неизвестно, какой будет наблюдаться шум в будущем, однако можно предполагать, что в среднем он будет равен 0. Поэтому значения будущих ошибок можно безболезненно заменить на 0. Фактически из уравнения просто удаляются все члены, которые связаны с ошибками из будущего:

$$\hat{y}_{T+1|T} = \hat{\alpha} + \hat{\phi}_1 y_T + \cdots + \hat{\phi}_p y_{T+1-p} + \hat{\theta}_1 \varepsilon_T + \cdots + \hat{\theta}_q \varepsilon_{T+1-q}.$$

В уравнении также присутствуют ошибки из прошлого. Их необходимо заменить на остатки модели в этих точках, потому они являются самыми лучшими оценками ошибок из имеющихся:

$$\hat{y}_{T+1|T} = \hat{\alpha} + \hat{\phi}_1 y_T + \cdots + \hat{\phi}_p y_{T+1-p} + \hat{\theta}_1 \hat{\varepsilon}_T + \cdots + \hat{\theta}_q \hat{\varepsilon}_{T+1-q}.$$

Если прогноз необходимо построить не на одну точку вперёд, а, например, на две, то в формуле появляется значение ряда из будущего y_{T+1} :

$$\hat{y}_{T+2|T} = \hat{\alpha} + \hat{\phi}_1 y_T + \cdots + \hat{\phi}_p y_{T+2-p} + \hat{\theta}_1 \hat{\varepsilon}_T + \cdots + \hat{\theta}_q \hat{\varepsilon}_{T+2-q}.$$

Его необходимо заменить на прогноз $\hat{y}_{T+1|T}$.

Прогноз реальной заработной платы в России на два года вперёд показан на рисунке 1.29. Использовалась модель $ARIMA(2, 0, 1) \times (2, 1, 2)$ и преобразование Бокса-Кокса. Полученный прогноз не выглядит тривиальным: он не является константой, не ведёт себя странно, не падает до нуля. Визуально этот прогноз можно одобрить.

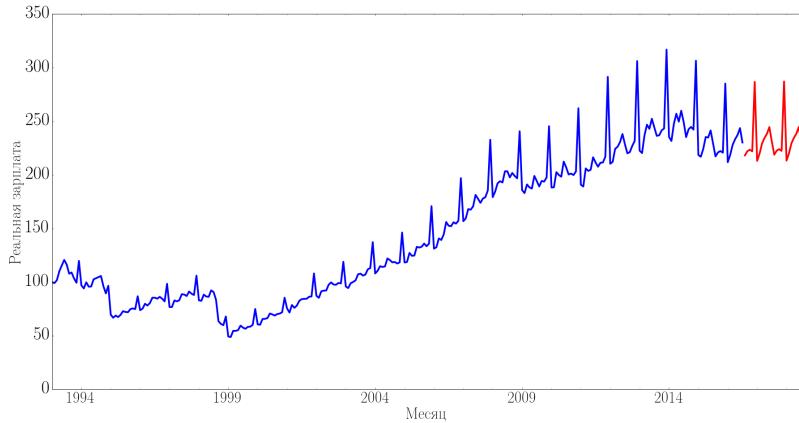


Рис. 1.29: Прогноз реальной заработной платы в России на два года вперёд (красным).

1.7. Анализ остатков

Анализ остатков — это техника, которая помогает понять, есть ли у прогнозирующей модели небольшие недостатки, которые можно устранить доработкой, или же фундаментальные проблемы.

Остатки — это разность между фактом и прогнозом:

$$\hat{\varepsilon}_t = y_t - \hat{y}_t.$$

Их можно вычислять двумя способами. Во-первых, прогнозы, которые участвуют в остатках, можно строить с фиксированной отсрочкой. Например, начиная с момента R прогноз всегда делается на одну точку вперёд, затем происходит переход в момент $R + 1$, получается новое истинное значение ряда, которое сравнивается с прогнозом, затем следующий прогноз делается ещё на одну точку вперёд, и так далее до самого конца ряда:

$$\hat{y}_{R+d|R}, \dots, \hat{y}_{T|T-d}.$$

Во-вторых, остатки можно строить с фиксированным концом истории при разных отсрочках. Например, берётся начальная часть ряда от 0 до $T - D$, и далее делаются прогнозы

$$\hat{y}_{T-D+1|T-D}, \dots, \hat{y}_{T|T-D},$$

полученные прогнозы сравниваются с истинными значениями ряда, и с их помощью вычисляются остатки.

В зависимости от задачи могут использоваться разные определения остатков, однако чаще используется первое. Остатки оценивают ошибку, то есть шумовую компоненту, которую наблюдать невозможно. При построении модели делаются предположения об этой шумовой компоненте, и логично, что свойства остатков должны согласовываться с выдвинутыми предположениями.

С предположениями о шумовой компоненте необходимо разобраться подробнее.

1.7.1. Несмешённость

Во-первых, остатки должны быть несмешёнными, то есть в среднем они должны быть равны нулю.

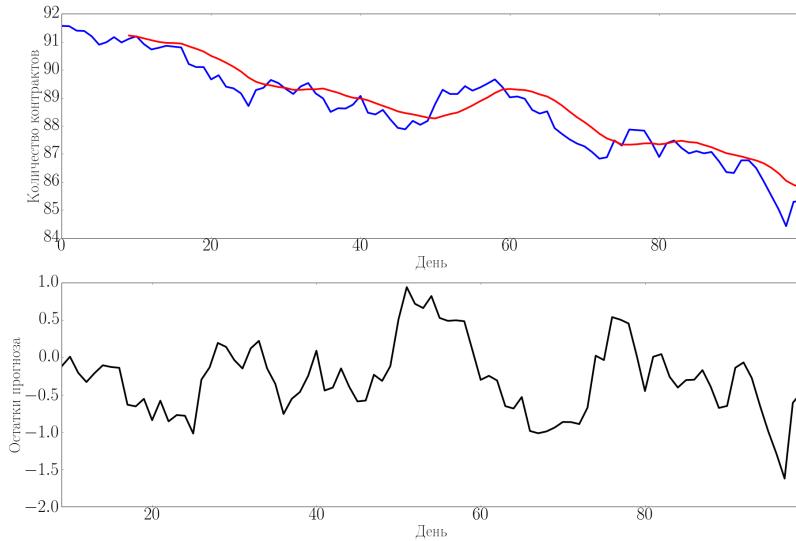


Рис. 1.30: Сверху — количество контрактов в сокровищнице США и прогноз, выполненный методом скользящего среднего, снизу — остатки модели.

На рисунке 1.30 показан прогноз, полученный методом скользящего среднего (по 10 предыдущим значениям). Это достаточно простой и грубый метод прогнозирования, и для рядов с ярко выраженным трендом он даёт плохие результаты. В данном случае видно, что прогноз завышает значения ряда, то есть отстаёт от понижающегося тренда. По этой причине остатки в среднем получаются отрицательными, что видно на нижнем графике на рисунке 1.30.

Гипотезу о несмещённости остатков $H_0: \varepsilon = 0$ можно формально проверить с помощью какого-либо стандартного одновыборочного критерия (например, критерия Стьюдента или Уилкоксона). Если выясняется, что остатки смещены, значит с моделью что-то не так. В этом случае рекомендуется провести визуальный анализ, чтобы посмотреть, почему прогнозы систематически завышаются или занижаются.

На самом деле, модель очень легко скорректировать в случае, если остатки имеют смещение. Достаточно вычислить среднее значение остатков, это и будет константой, на которую необходимо скорректировать все прогнозы, чтобы остатки стали несмещёнными. После этого преобразования прогнозирующая модель улучшится.

1.7.2. Стационарность

Ещё одно свойство, наличие которого предполагается у ошибок, — это стационарность, то есть отсутствие зависимости от времени. Таким образом, остатки во времени должны быть распределены примерно одинаково.

На рисунке 1.31 показан прогноз, построенный так называемым наивным сезонным методом. То есть каждый январь значение ряда предсказывается равным предыдущему январю и т.д. Поскольку в ряде присутствует ярко выраженный тренд, то такой прогноз будет некачественным, а остатки — смещёнными. В таком случае необходимо провести корректировку на смещение. По остаткам скорректированного несмещённого прогноза (рис. 1.31, снизу) видно, что в модели всё ещё есть проблема. Остатки ведут себя систематически: сначала они в основном отрицательные, затем в течение какого-то времени — положительные, в конце ряда — снова преимущественно отрицательные.

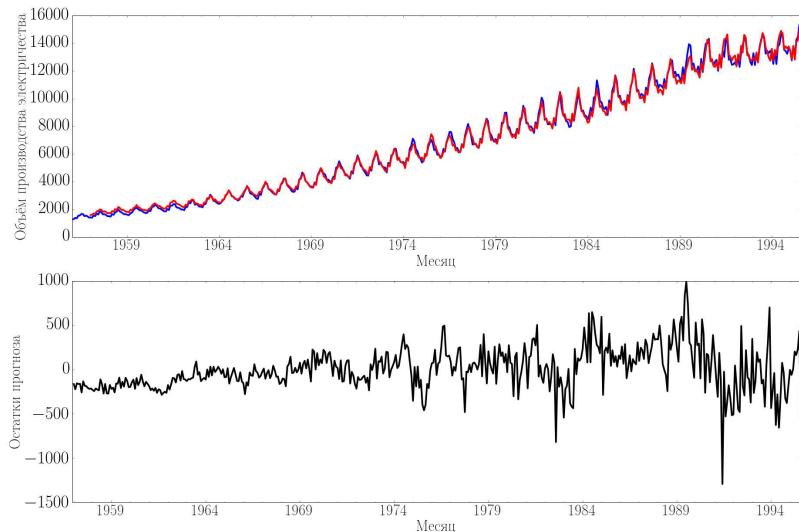


Рис. 1.31: Сверху — количество произведённого в Австралии электричества и прогноз, выполненный "наивным сезонным методом" с корректировкой на смещение, снизу — остатки модели.

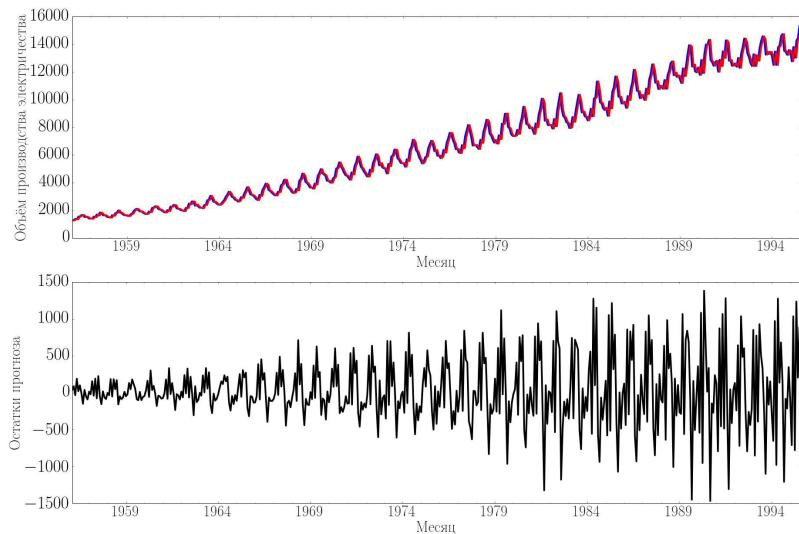


Рис. 1.32: Сверху — количество произведённого в Австралии электричества и наивный прогноз, снизу — остатки модели.

Результат применения наивного прогноза (в каждой точке значение прогнозируется предыдущим значением этого же ряда) показан на рисунке 1.32. Как и в предыдущем случае, остатки модели нестационарны: дисперсия меняется во времени.

Формально гипотезу о стационарности можно проверить с помощью критерия Дики-Фуллера. Если стационарность отсутствует, то модель неодинаково точна в разные периоды времени. Необходимо провести визуальный анализ, чтобы понять, что с моделью не так, и почему прогнозы в разные периоды времени систематически имеют разную ошибку.

1.7.3. Неавтокоррелированность

Ещё одно желаемое свойство остатков — это неавтокоррелированность, то есть отсутствие зависимости от предыдущих наблюдений.

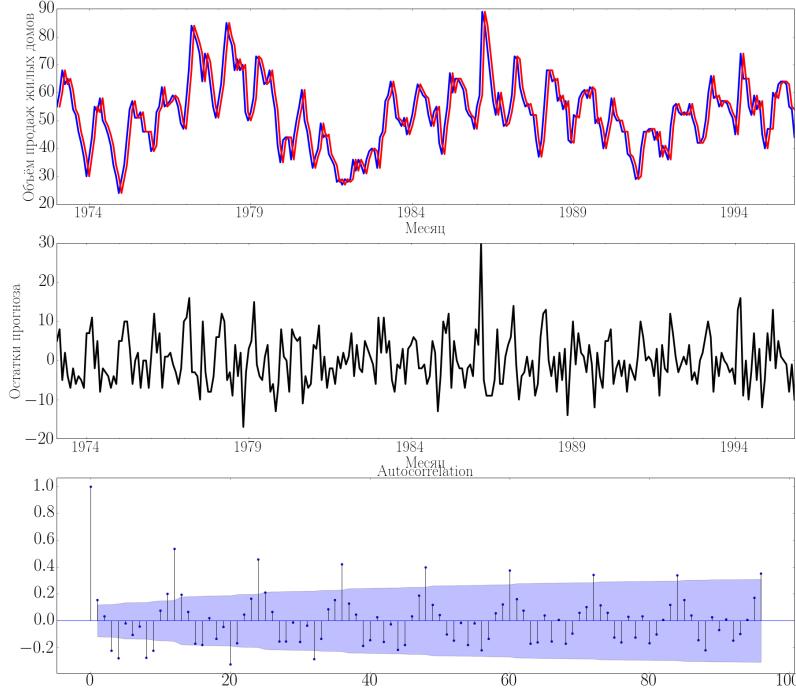


Рис. 1.33: Сверху — продажи недвижимости в США и прогноз, выполненный наивным методом, по центру — остатки модели, снизу — автокорреляционная функция остатков.

На рисунке 1.33 показан результат прогнозирования наивным методом продаж недвижимости в США. Наивный метод работает очень плохо при работе с рядами с ярко выраженной сезонностью. Это хорошо видно на графике автокорреляционной функции остатков прогноза. Она имеет очень значимые и ярко выраженные пики на лагах, кратных сезонному периоду.

Гипотезу о неавтокоррелированности можно проверить по коррелограмме, а также с помощью Q-критерия Льюнга-Бокса (таблица 1.3). Этот критерий позволяет проверить гипотезу о равенстве нулю одновременно нескольких автокорреляций при разных лагах (с лага 1 по лаг Q). Параметр Q можно выбирать, например, перебором, а можно пользоваться значением по умолчанию, использующемся в функции, которая производит оценку модели ARIMA.

ряд ошибок прогноза:	$\varepsilon^T = \varepsilon_1, \dots, \varepsilon_T$
нулевая гипотеза:	$H_0: r_1 = \dots = r_Q = 0$
альтернатива:	$H_1: H_0$ неверна
статистика:	$Q(\varepsilon^T) = T(T+2) \sum_{\tau=1}^Q \frac{r_\tau^2}{T-\tau}$
нулевое распределение:	$Q(\varepsilon^T) \sim \chi_{Q-K}^2$ при H_0
K —	число настраиваемых параметров модели

Таблица 1.3: Описание Q-критерия Льюнга-Бокса

Автокоррелированность остатков — признак того, что в данных присутствует информация, которая не вошла в модель. Если в остатках есть структура, то можно попытаться её внести в модель явным образом. Скорректированная модель будет лучше, а её остатки будут больше похожи на белый шум. Однако это можно сделать далеко не всегда — возможности модели класса ARIMA не безграничны, и с помощью таких моделей нельзя учесть всю структуру ряда. Таким образом, автокоррелированность остатков только указывает на потенциальную возможность улучшить модель, и не факт, что улучшения можно добиться на практике с помощью рассматриваемого класса моделей.

1.8. Пример

1.9. Регрессионный подход к прогнозированию

1.9.1. Праздники

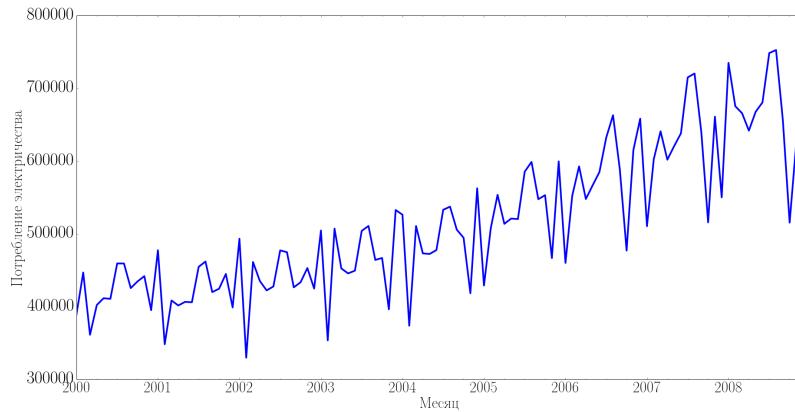


Рис. 1.34: Ежемесячное потребление электричества в Турции

На рисунке 1.34 представлены ежемесячные данные о потреблении электричества в Турции. На графике очень хорошо видна годовая сезонность, а также странные падения. Эти падения соответствуют месяцам, на которые выпадают праздники по исламскому календарю. Этот календарь примерно на 11 дней короче, чем григорианский. По этой причине праздники, которые определяются по исламскому календарю всё время попадают в разные места стандартного григорианского года, и их не получается учесть с помощью стандартной сезонности с периодом 12. Это плохая новость.

Есть и хорошая новость, которая заключается в том, что для каждого года точно известно, на какой месяц приходится такой праздник. На основании этого можно сформировать бинарный признак, в котором 1 будет соответствовать месяцам с праздником, 0 — остальным. Такие признаки можно попытаться встроить в модель типа ARIMA.

1.9.2. ARIMAX

Пусть считается, что значение ряда y в момент времени t задаётся следующим образом:

$$y_t = \sum_{j=1}^k \beta_j x_{jt} + z_t,$$

то есть фактически y задаётся линейной регрессией, но остатки этой регрессии прогнозируются при помощи модели ARIMA:

$$\begin{aligned} z_t = & \alpha + \phi_1 z_{t-1} + \cdots + \phi_p z_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \\ & + \phi_S z_{t-S} + \cdots + \phi_{PS} z_{t-PS} + \theta_S \varepsilon_{t-S} + \cdots + \theta_{PS} \varepsilon_{t-PS} + \varepsilon_t. \end{aligned}$$

Такая модель, которая комбинирует линейную регрессию и ARIMA называется regARIMA или ARIMAX.

1.9.3. Сложная сезонность

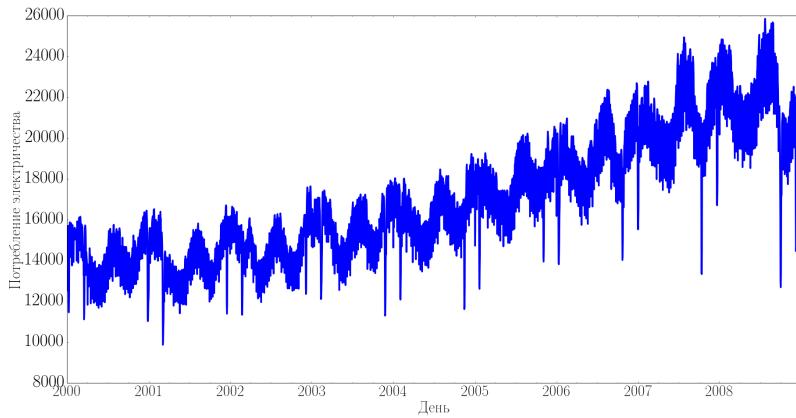


Рис. 1.35: Ежедневное потребление электричества в Турции

Описанная выше модель очень полезна при работе с рядами со сложной сезонностью. Например, если рассмотреть потребление электричества не по месяцам, а по дням (рис. 1.35), то в этом ряду будет недельная сезонность, годовая и, опять же, праздники по исламскому календарю.

К сожалению, модели класса ARIMA дают плохие результаты при работе с рядами со сложной сезонностью. Возникает несколько проблем. Во-первых, при длинных сезонных периодах в модели ARIMA становится слишком много параметров, и их невозможно оценить по ряду.

Во-вторых, модель ARIMA явно задаёт значение ряда как функцию от его значения, например, один сезонный период назад. Странно ожидать, что при работе с подневным рядом его значение будет определяться значением в эту же дату прошлого года. Скорее всего, значения ряда в некой окрестности текущей даты похоже на его значения в этой же самой окрестности этой же даты год назад.

Наконец, довольно большую проблему представляет то, что длина года нецелая. Она равна не в точности 365 дням, а 365.25. Если в ряду собраны недельные данные, то длина года в неделях также не 52 недели ровно, а 52.18. С этой проблемой ARIMA не может работать никак, в ряде нет измерения с индексом 52.18, которое можно было бы подставить в модель.

Решить эту проблему можно в рамках регрессионного подхода. Для модели ARIMA можно оставить самый короткий из имеющихся сезонных периодов. Все остальные периоды будут учитываться с помощью регрессии на специально построенные признаки. В качестве признаков будут выступать фурье-гармоники с периодами, пропорциональными длине сезонных периодов (например, 365.25, 365.25/2, 365.25/3 и т.д.). Какое-то количество таких гармоник необходимо подставить как регрессионную компоненту в регрессионную ARIMA.

1.9.4. Регрессионные признаки

Вообще говоря, можно придумать много вещей, которые можно подставлять в регрессионную компоненту:

- гармоники по длинным периодам сезонности;
- для коротких периодов сезонности можно использовать индикаторы (например, дни недели можно явно задать как индикатор понедельника, индикатор вторника, и т. д.) и в явном виде подставить их в регрессионную ARIMA;
- индикаторы праздников;
- полезными часто оказываются индикаторы пред- и постпраздничных дней;
- тренды (линейный, квадратичный и т. д.);
- скользящие средние ряда за предыдущие периоды (например, в каждой точке вычисляется среднее за прошлый месяц или прошлую неделю, и такой признак подставляется в регрессионную ARIMA).

Часто оказывается, что если хорошо подобрать признаки, и построить на них регрессию, то добавлять поверх неё ARIMA уже не нужно. Выигрыш в качестве при добавлении авторегрессионной добавки оказывается практически незначимым.

1.9.5. Массовое прогнозирование

Специфические методы прогнозирования рядов хороши, когда работа производится не более чем с десятками временных рядов, которые постоянно прогнозируются. На каждый из этих рядов можно посмотреть глазами, тщательно подобрать хорошую модель, проанализировать остатки, при необходимости перестроить модель и т.д. Это ручной труд, и часто нет возможности для такого ручного труда. Нет такой возможности в задачах массового прогнозирования.

Пусть необходимо спрогнозировать дневные продажи товаров в магазинах. В распоряжении имеются данные о продажах товаров, их остатках в магазинах, ценах, скидках, промоакциях, иерархии товаров, иерархии и расположение торговых точек. Стоит задача построить прогнозы продаж всех товаров во всех магазинах. Однако временных рядов слишком много, чтобы их можно было спрогнозировать вручную.

На помощь может прийти регрессионный подход. Если хорошо сконструировать признаки, которые будут содержать всю имеющуюся информацию, и использовать на них какую-то регрессионную модель (не обязательно линейную), то уже таким способом можно получить достаточно хорошее решение.

1.9.6. Резюме

Оказалось, что задачу прогнозирования временных рядов можно свести к задаче обучения с учителем, не применяя специфические методы. Тем не менее, важно помнить две вещи. Во-первых, для того, чтобы регрессионный подход работал, необходимо явно учитывать временную природу данных при конструировании признаков. Во-вторых, даже если получилось построить хорошую регрессионную модель, стоит попробовать поверх неё наложить авторегрессию или какую-либо ещё модель временных рядов, о которых не упоминалось в этом курсе. Возможно, это поможет улучшить качество прогнозов.

Урок 3

Компьютерное зрение

3.1. Компьютерное зрение

Компьютерное зрение — это область науки, которая занимается разнообразными задачами, связанными с анализом изображений и видео. Можно считать, что во всех них требуется ответить на вопрос, что изображено на картинке. Несмотря на кажущуюся тривиальность вопроса, ответить на него не так просто.

3.1.1. Классификация сцены



Рис. 3.1

На рисунке 3.1 показана картинка. Чтобы ответить на вопрос, что на ней изображено, можно описывать сцену в целом. Понятно, что картинка сделана на улице (вне помещения), где-то в азиатской стране (кто-то может узнать площадь Тяньаньмэнь в Пекине).

Другой подход — выделять отдельные объекты на изображении (рис. 3.2). На картинке видно автобус, портрет, крышу, небо и т.д.

Можно пойти дальше и говорить про физические свойства отдельных объектов. Например, крыша — наклонная, автобус едет, и он твёрдый, на стене висит изображение Мао Цзедуна, ветер дует справа налево (это можно определить по движению флага).

Из примера выше можно заключить, что для ответа на вопрос, что изображено на рисунке, используется весь жизненный опыт. Например, знание о том, что существует ветер (на картинке его нельзя увидеть в явном виде), что такое транспорт. Для того чтобы ответить на определённые вопросы, необходимо знать историю Китая. Соответственно задача заключается не в том, чтобы смотреть на пиксели, а в использовании знаний.



Рис. 3.2

3.1.2. Внутриклассовая изменчивость



Рис. 3.3

Другой пример. На вопрос, что такое стул, можно ответить первое, что придёт в голову. Например, стул — это нечто с четырьмя ножками и спинкой. Фотография на рисунке 3.3 показывает, что не все стулья подходят под это определение.

Стул достаточно сложно описать в терминах форм. Стул — это некое концептуальное понятие, то, на чём сидят. Можно представить, как сложно объяснить это понятие инопланетному существу, которое не знает даже, что такое сидеть и не умеет это делать. Прежде чем научить существо находить на картинках стул, было бы неплохо, чтобы оно поняло концепцию "сидеть". Абсолютно то же самое происходит, когда компьютер учат распознавать изображения. В идеале, чтобы он отвечал на вопросы про стулья так же хорошо, как человек, ему нужно понимать концепцию "сидеть".

3.1.3. Искусственный интеллект

В науке об искусственном интеллекте существует понятие "ИИ-сложные задачи". Это класс задач, решение которых эквивалентно созданию искусственного интеллекта. Считается, что задача компьютерного зрения в общей постановке (ответ на вопрос, что изображено на картинке, и на все вопросы про это изображение) является ИИ-сложной. Выше было продемонстрировано, что действительно, для ответа на вопрос об изображении не просто смотреть, а использовать весь свой жизненный опыт, образование, а иногда и интуицию.

К сожалению, искусственный интеллект до сих пор не создан. Поэтому наука о компьютерном зрении решает только определённые подзадачи, речь о которых пойдёт далее.

3.2. Задачи компьютерного зрения

Далее будут приведены примеры нескольких задач, которые решаются с применением технологии компьютерного зрения.

3.2.1. Поиск по изображению

Первый пример — поиск изображений в интернете. Сейчас существует несколько сервисов, которые позволяют искать картинки. Изначально для поиска использовались текстовые запросы. Некоторое время назад в части из таких сервисов появилась возможность поиска по загруженному изображению. От пользователя требуется загрузить картинку, а сервис будет искать похожие на неё изображения в интернете.

На самом деле, это работает следующим образом. Сначала индексируются изображения из интернета. Для них строятся некоторые цифровые представления, из которых формируется структура данных, по которой можно быстро производить поиск. Для пользовательской картинки также используется цифровое представление, по которому в сформированной структуре данных ищутся дубликаты или похожие картинки.

Данная задача является сложной в структурном смысле. В интернет загружены миллиарды изображений, и использование сложных методов сравнения невозможно, потому что необходимо достигать высокой производительности.

3.2.2. Распознавание текста

Следующий пример — это технология распознавания теста. Необходимо найти изображение текста на картинке и представить его в виде текстовых данных, с которыми можно будет работать, например, в редакторе. Эта технология используется в разнообразных приложениях. В частности, это удобный способ вводить текст в онлайн-переводчик. Достаточно сфотографировать этикетку, текст на ней будет распознан, и переводчик выполнит перевод (рис. 3.4).



Рис. 3.4: Пример использования технологии распознавания речи в приложении-переводчике

3.2.3. Биометрия

Ещё одна задача, решаемая в рамках науки о компьютерном зрении, — это биометрия, распознавание людей. Для этого может использоваться изображение лица, радужная оболочка глаза, отпечатки пальцев. Однако в основном компьютерное зрение занимается распознаванием лиц. С каждым годом эта технология работает всё лучше и лучше, и находит широкое применение.

3.2.4. Видеоаналитика

В мире устанавливается всё больше камер: на дорогах для регистрации движения автомобилей или в общественных местах для отслеживания потоков людей и детектирования аномалий (например, оставленные вещи, нелегальные действия). Как следствие, возникает задача анализа огромного потока появившейся информации. Компьютерное зрение помогает в решении этой задачи. Оно позволяет детектировать номер автомобиля, его марку, нарушает ли он правила дорожного движения.

3.2.5. Анализ спутниковых снимков

В данный момент накоплен огромный массив спутниковых снимков. Используя эти данные, можно решать разнообразные задачи: улучшать карты, детектировать лесные пожары и другие проблемы, которые видны со спутника. Технологии компьютерного зрения шагнули в последнее время далеко вперёд, и с их использованием автоматизируется всё больше ручной работы в этой области.

3.2.6. Графические редакторы

Компьютерное зрение позволяет не только распознавать, что изображено на картинке. Также оно позволяет изменять и улучшать изображения. Таким образом, всё, что можно сделать с помощью графического редактора, относится к технологии компьютерного зрения.

3.2.7. 3D анализ

3D-реконструкция — ещё один пример задачи, решаемой с помощью компьютерного зрения. Используя множество изображений, сделанных в данном городе, можно восстановить форму зданий.

3.2.8. Компьютерное зрение и авто

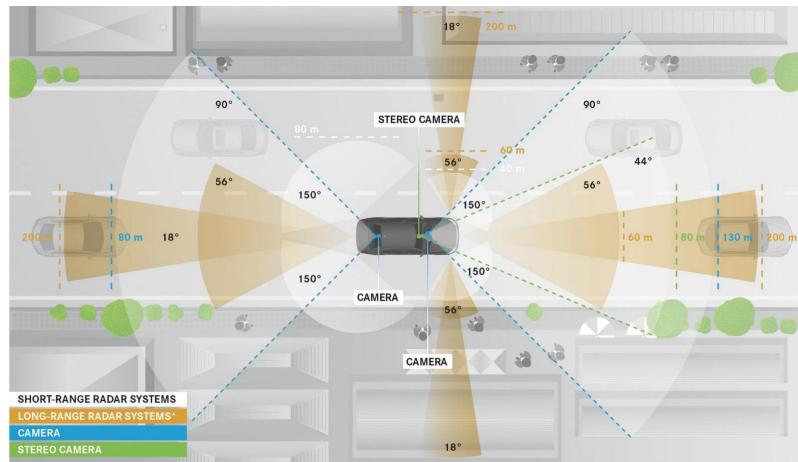


Рис. 3.5

Современные автомобили оснащены огромным количеством датчиков: несколько видеокамер, радары, стереокамера (рис. 3.5). Методы компьютерного зрения помогают анализировать информацию, получаемую с этих датчиков. С использованием этих методов созданы системы предотвращения ДТП, столкновения с пешеходами и предупреждения водителя о разнообразных препятствиях.

Кроме того, сейчас активно развивается область автопилотируемых автомобилей. В них технологии компьютерного зрения используются для ориентирования в пространстве.

3.3. "Низкоуровневое" зрение

Методы компьютерного зрения используются для решения задач, которые условно можно разделить на простые и сложные. Сложные задачи отвечают на вопросы, какой объект изображён на картинке, к какому

классу он относится. Для решения этих задач чаще всего используются методы машинного обучения. При решении простых задач производятся манипуляции непосредственно с пикселями, используются эвристики, а методы машинного обучения, как правило, не применяются. Этот раздел посвящён задачам "простого" компьютерного зрения, "низкоуровневому" зрению. Стоит отметить, что они часто используются как составная часть более сложных задач распознавания. Например, предобработка картинки позволяет алгоритмам машинного обучения лучше распознавать, что на ней изображено.

3.3.1. Библиотеки

OpenCV — это самая популярная библиотека, используемая для решения задач "низкоуровневого" компьютерного зрения. В ней содержится огромное количество алгоритмов, есть интерфейсы для языков C++ и python. Другая известная библиотека — skimage, она активно используется в скриптах на языке python. В этом курсе для демонстрации примеров будет использоваться библиотека opencv.

3.3.2. Представление изображений



0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

Рис. 3.6: Представление изображения в памяти компьютера

Прежде всего необходимо разобраться, как картинки представляются в памяти компьютера. Если человек видит некие цельные картины и объекты, то в памяти компьютера это числа (рис. 3.6). Изображение состоит из пикселей. Если оно не цветное, то каждый пиксель описывается яркостью (для однобайтовых изображений яркость принимает значения от 0 до 255). Если в картинке есть цвет, описание пикселя состоит из трёх чисел. Самое простое и интуитивно понятное из них — это яркости изображения в красном, зелёном и синем канале. Существуют также и другие представления цвета, о них речь будет идти позже.

3.3.3. Арифметические операции

Итак, картинки — это матрицы чисел. В случае чёрно-белых картинок это матрицы размера высота на ширину картинки. В случае цветных изображений у матрицы появляется ещё одна размерность, чаще всего она равна трём.

В opencv используется такое же представление матриц, как в библиотеке numpy. Это означает, что для них можно использовать стандартные арифметические операции, например, сложение. Однако не всё так просто: сложение матриц в numpy не учитывает переполнение. Для изображений переполнение — это нелогичная операция. Если при сложении двух картинок яркость где-то превысила 255, то, как правило, она должна оставаться равной 255, а не превратиться в 4. Пример ниже показывает, как отличается сложение в numpy и opencv.

```
import numpy as np
import cv2
x = np.uint8([250])
y = np.uint8([10])
print cv2.add(x,y)
[[255]]
```

```
x + y  
[4]
```



Рис. 3.7

Работу этих операций можно продемонстрировать на примере рисунка 3.7. Сначала изображение нужно сделать серым.

```
img1 = cv2.imread('imgs/lenna.jpg')  
gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
```



Рис. 3.8

Команда `cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)` будет неоднократно встречаться в дальнейшем. Она используется для преобразования цветовых пространств, в том числе, из RGB в серые картинки. После превращения картинки в серую, можно прибавить к ней какое-то число (рис. 3.8):

```
gray_add = cv2.add(gray, 50)
```

Такое преобразование эквивалентно увеличению яркости картинки. Можно не прибавлять, а умножать на некий коэффициент (рис. 3.8):

```
gray_mul = cv2.multiply(gray, 1.3)
```

Умножение картинки эквивалентно увеличению её контрастности. Можно попробовать использовать больший коэффициент (рис. 3.9а):



(a)



(b)

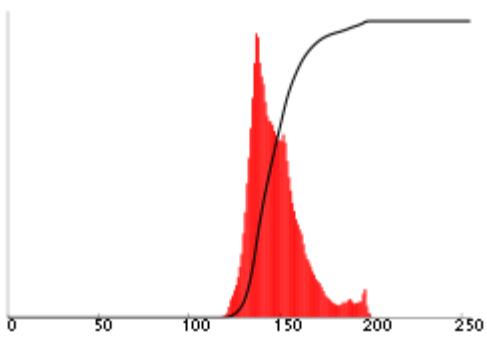
Рис. 3.9: Результат умножения картинки на число

```
gray_mul = cv2.multiply(gray, 1.8)
```

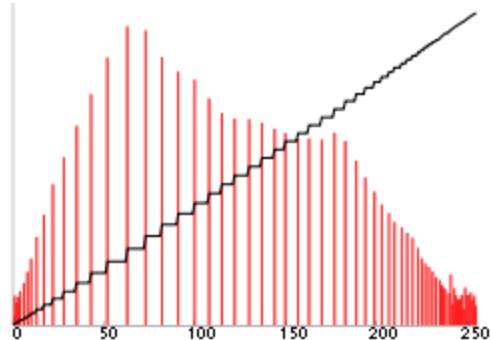
Ранее был рассмотрен пример того, как меняется изображение после сложения или умножения на некоторое число. Именно так и работают алгоритмы изменения яркости и контраста во многих популярных графических редакторах и мобильных приложениях. Однако для этой цели можно использовать и более сложные функции.

3.3.4. Эквилизация гистограммы

Пример такого подхода — это эквилизация гистограммы. В данном случае гистограмма — это представление картинки, показывающее, сколько в ней пикселей той или иной яркости. На рисунке 3.10а показана гистограмма какого-то изображения. Чёрная линия — это кумулятивная гистограмма, отвечающая на вопрос, у какого количества пикселей яркость меньше значения x .



(a)



(b)

Рис. 3.10: Гистограмма до эквилизации (а) и после (б)

В результате эквилизации гистограмма картинки растягивается таким образом, чтобы кумулятивная гистограмма была близка к линейной функции. Результат эквилизации изображения показан на рисунке 3.11. Он получен с помощью следующей функции:

```
equ = cv2.equalizeHist(gray)
```

Видно, что в результате контраст изображения улучшился: тёмные области стали темнее, яркие — светлее.

Чтобы произвести эквилизацию гистограммы, нужно применить некое преобразование к яркости пикселей. Конкретный вид преобразования можно попробовать получить самостоятельно, это хорошая задачка на алгоритмы.



Рис. 3.11: Результат эквилилизации гистограммы

3.3.5. Блендинг

Блендинг — это ещё один пример применения простых арифметических операций к картинкам. Ставится задача скомбинировать два изображения. Можно попробовать их сложить. Но в таком случае, если объекты наложатся друг на друга, то получится каша.



Рис. 3.12: Совмещение двух изображений

Пусть для одной картинки известно, где расположен объект, а всё остальное пространство занимает фон. Тогда можно помещать второй объект туда, где находится фон. В месте, где первый объект накрывается вторым, будет также использоваться второй объект.

Такое объединение достаточно требовательно к качеству вырезания картинки. Если по краям неаккуратно обрезан фон, то по краям будет видна некрасивая белая полоса (рис. 3.12a).

Кажется, что научиться аккуратно вырезать объект из фона — сложная задача. Это так, потому что фон неоднородный, и недостаточно просто выбросить белые пиксели. Можно воспользоваться хитрым алгоритмом смещивания двух картинок, и построить маску таким образом, что её значение будет тем больше, чем дальше пиксель от белого. В таком случае там, где на исходном изображении располагаются белые пятна, будут браться пиксели со второго изображения, и неаккуратное вырезание объекта будет не так заметно. На рис. 3.12 показано, как такое простое преобразование помогает избавиться от этой проблемы.

Существуют более сложные алгоритмы блендинга. В случаях, когда требуется скопировать объект с неоднородным фоном и вставить его в другое изображение (рис. 3.13), простые методы, смешивающие цвета не помогают. Существуют более хитрые методы, использующие оптимизацию, чтобы определить, где находится



Рис. 3.13: Источник: Pérez, Patrick, Michel Gangnet, and Andrew Blake. "Poisson image editing."

объект, а где — фон. Затем свойства объекта переносятся без изменений, а свойства фона берутся с картинки, на которую вставляется объект.

3.3.6. Цветовые пространства

Ранее уже упоминались цветовые пространства. Самое интуитивно понятное и популярное из них — представление цвета в каналах красного, зелёного и синего, RGB.

HSV

Другой пример цветового пространства — это HSV (рис. 3.14). Компонентами этого пространства являются тон (hue), насыщенность (saturation) и значение (value). Это пространство позволяет манипулировать цветом и его насыщенностью по отдельности. Тон обозначает цвет пикселя, он закодирован числом от 0 до 360, как угол на цилиндре (рис. 3.14). Насыщенность принимает значение 0, если картинка серая.

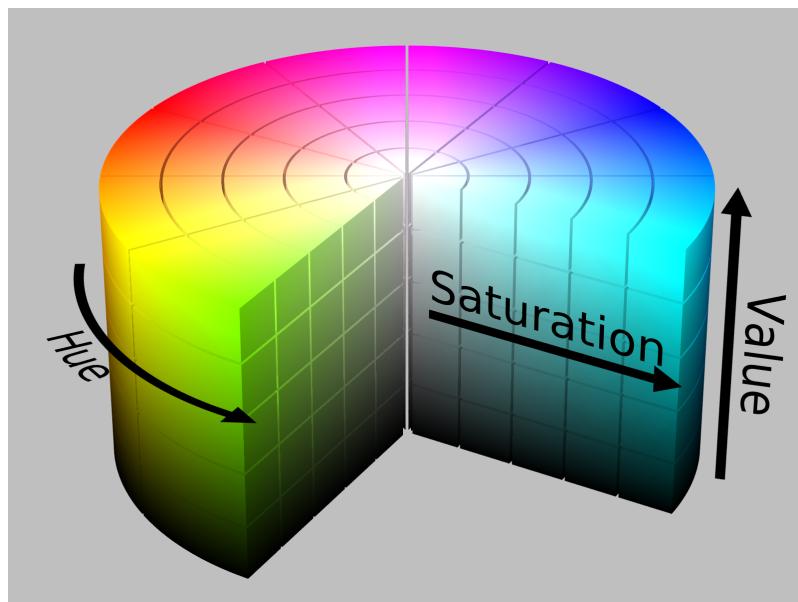


Рис. 3.14: Цветовое пространство HSV

На рисунке 3.15 показано, как меняется картинка из примера при умножении канала насыщенности на 1.5 в цветовом пространстве HSV. В результате цвета стали более сочными и насыщенными. Это выполняется при помощи следующего кода:

```
hsv = cv2.cvtColor(img1,
```

```

cv2.COLOR_BGR2HSV)
hsv[:, :, 1] = cv2.multiply(hsv[:, :, 1], 1.5)
result = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

```



(a)

(b)

Рис. 3.15: Фото до (а) и после (б) умножения канала насыщенности на 1.5 в цветовом пространстве HSV

3.3.7. Каскады Хаара — детектор лиц

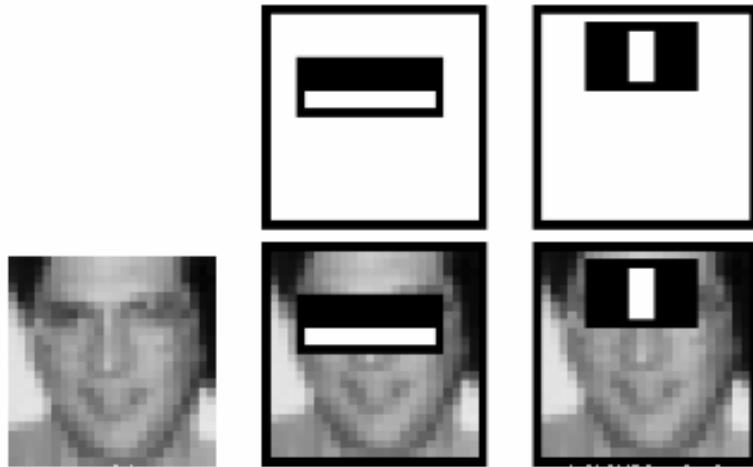


Рис. 3.16: Каскады Хаара

Одна из задач, решаемых наукой о компьютерном зрении, — это детекция лиц. Один из первых удачных методов решения — это каскады Хаара. Применяя этот метод, из картинки можно вычленять достаточно простые признаки. Для этого необходимо использовать несколько прямоугольников (рис. 3.16). Пиксели, попадающие в белый прямоугольник берутся со знаком плюс, в чёрный — со знаком минус. Все значения суммируются, и получается одно число. Выбор прямоугольников и коэффициентов для них осуществляется с помощью алгоритма AdaBoost. У лица имеются некоторые паттерны, и в итоге каскад такого рода фильтров показывает, есть ли внутри него лицо, или нет.

Сейчас созданы методы детекции лиц, превосходящие по качеству каскады Хаара. Тем не менее этот подход достаточно простой, и его легко найти готовым к использованию. Если не требуется решать задачу с высоким качеством, а получить детектор нужно быстро и просто, каскады Хаара из библиотеки opencv — это отличный вариант.

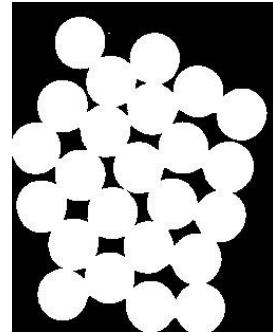
3.3.8. Сегментация

Задачу сегментации также можно достаточно просто решить. Один из способов — отрезать пиксели выше какого-то порога и назначить их объекту, а пиксели ниже порога — фону. Пример показан на рисунке 3.20. На этом изображении требуется разделить монеты и фон. Видно, что монеты намного темнее, поэтому достаточно подобрать такую границу, чтобы они все оказались ниже. Код из opencv, который позволяет это сделать:

```
img = cv2.imread('coins.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```



(a)



(b)

Рис. 3.17: Сегментация изображения

3.4. Линейная фильтрация изображений

Важный класс преобразования изображений — это линейные фильтры. С их помощью решаются задачи поиска границ, уголков, удаления шумов.

3.4.1. Скользящее среднее — свёртка

$$\frac{1}{9} \begin{matrix} h \\ \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \end{matrix}$$

Рис. 3.18

Проще всего объяснить, что такое линейная фильтрация, на примере. Пусть требуется подсчитать среднее в окне 3×3 для каждого пикселя. Вычисление среднего можно записать в следующем виде:

$$g[n, m] = \frac{1}{9} \sum_{k=n-1}^{n+1} \sum_{l=m-1}^{m+1} f[k, l] = \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[n-k, m-l]$$

Переписав его в следующем виде, можно получить выражение для свёртки:

$$(f \cdot h)[m, n] = \frac{1}{9} \sum_{k, l} h[m - k, n - l],$$

где f — это изображение (двухмерная функция, характеризующая картинку), k, l — координаты пикселя, f — яркость пикселя, h — ядро свёртки (матрица 3×3 , состоящая из единиц, рис. 3.18). Если ядро свёртки — матрица, как на рисунке 3.18, то свёртка — это скользящее среднее.

В opencv произвести такую свёртку можно следующим образом:

```

kernel = np.array([[1,1,1],[1,1,1],[1,1,1]],np.float32) / 9
dst = cv2.filter2D(img1,-1,kernel)

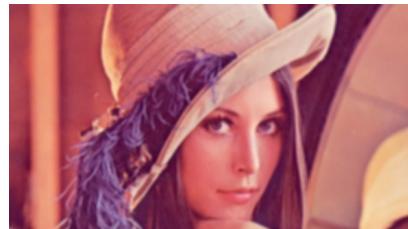
```

Картинка при этом становится более размытой (рис. 3.19). Также размытие изображения можно получать при помощи свёртки с гауссовской функцией (рис. 3.19):

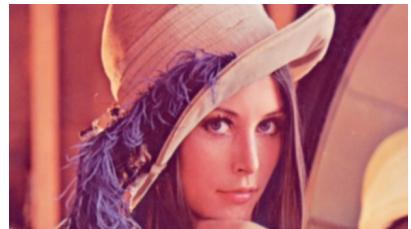
```
cv2.GaussianBlur(img1,(5,5), 0)
```



(a)



(b)



(c)

Рис. 3.19: (а) — фото до, (б) — после применения скользящего среднего, (с) — после применения Гауссовского размытия

3.4.2. Детекция границ

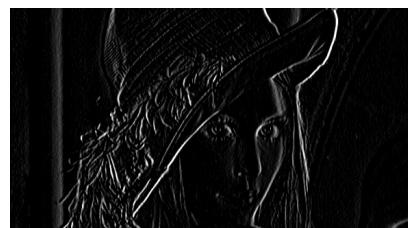
Свёртки также можно применять для детекции границ. С помощью свёрток 3.20a, 3.20b можно получить вертикальные и горизонтальные границы изображения (рис. 3.21a, 3.21b). Если объединить результаты этих двух свёрток, можно получить все границы (рис. 3.21c).

-1	-1	-1
0	0	0
1	1	1

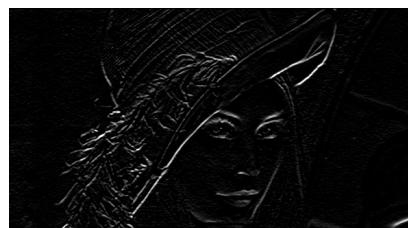
(a)

-1	0	1
-1	0	1
-1	0	1

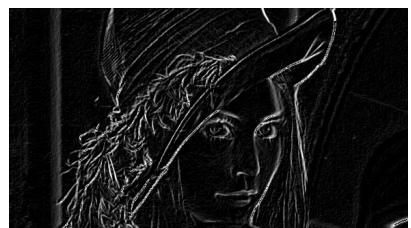
(b)



(a)



(b)



(c)

Рис. 3.21: Границы изображения, полученные с помощью применения свёртки. (а) — горизонтальные, (б) — вертикальные, (с) — все границы, полученные объединением результатов двух свёрток.

Такие ядра являются частью преобразования Превитта, их использование — это самый простой способ найти границы изображения.

На самом деле, существует много способов определения границ. Каждый из них применяется в разных условиях, и, в зависимости от задачи, необходимо использовать тот или иной способ.

3.4.3. Корреляция

Другой пример линейного преобразования — это корреляция. Она очень похожа на свёртку, но записывается немного в другом виде:

$$r_{f,g}[k,l] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n,m]g * [n-k, m-l] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n+k, m+l]g * [n, m], \quad k, l \in \mathbb{Z}.$$

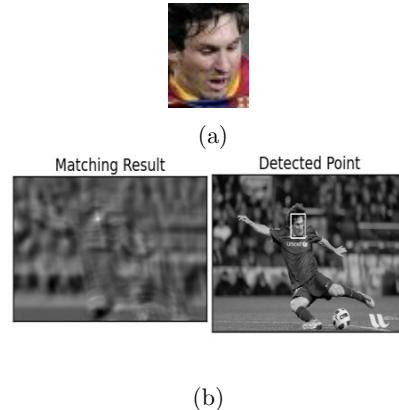


Рис. 3.22: (a) — изображение, которое требуется найти. (b) — слева — применение корреляции к картинке для поиска изображения, справа — картинка, содержащая искомое изображение

В отличии от свёртки, корреляция используется, чтобы показать меру похожести двух изображений. Это может быть использовано для поиска объектов. Например, требуется найти лицо футболиста (рис. 3.22a). На рисунке 3.22b слева показан результат применения корреляции для поиска лица. Белое пятно — это место, где оно найдено. Корреляцию можно использовать с различными параметрами: нормировать, применять её различные вариации.

Итак, корреляция — это очень простой способ поиска объектов на изображении, если имеется их точные копии.

3.4.4. Резюме

В этой части шла речь о линейной фильтрации. Позднее будет показано, что самые важные компоненты глубоких нейронных сетей состоят из свёрток, одного из способов линейной фильтрации. По этой причине важно получить понимание того, что такое свёртка, и для чего она используется. Далее речь пойдёт о том, как свёрточные нейронные сети помогают решать более сложные задачи компьютерного зрения.

Урок 4

Нейронные сети для анализа изображений

4.1. Классификация изображений

Классификация изображений — это задача, в рамках которой картинку необходимо отнести к одной из нескольких категорий. Примером может служить бинарная классификация изображений на сделанные в помещении и вне помещения, или многоклассовая классификация изображений собак на различные породы.

4.1.1. База ImageNet

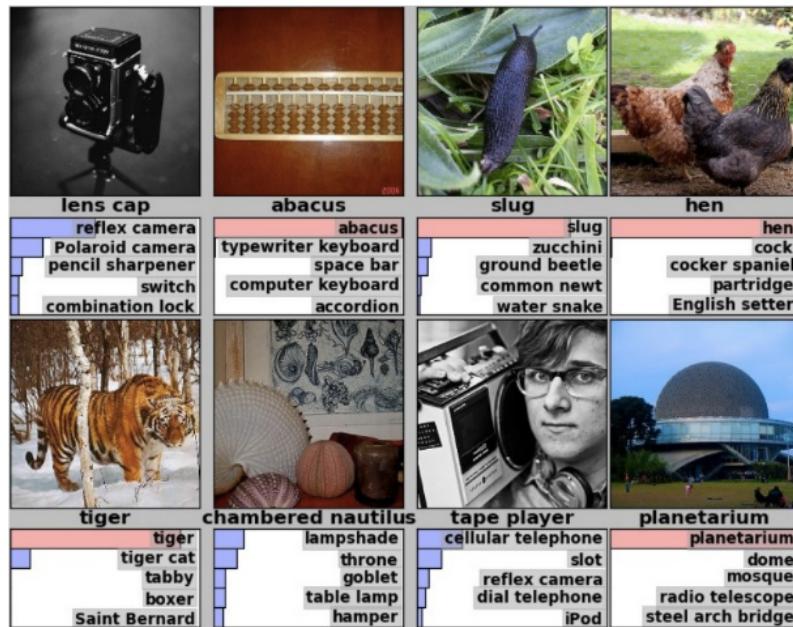


Рис. 4.1: Примеры изображений из базы ImageNet

База ImageNet содержит 10 млн изображений из интернета. Содержащиеся в ней картинки достаточно разнообразные, и все они размечены на принадлежность к тому или иному классу. Примеры изображений из базы показаны на рисунке 4.1.

Стоит отметить, что в базе встречаются ошибки и спорные случаи. Например, изображение на рисунке 4.2 отмечено в базе как "вишня". Если отнести его, например, к классу "дальматинец", то ответ будет считаться неправильным.

На базе ImageNet периодически проводятся соревнования по классификации изображений. Используется 1 млн изображений, каждое из которых требуется отнести к одному из 1000 классов. Можно давать больше



Рис. 4.2: Пример спорной классификации изображения в базе ImageNet, оно отмечено как "вишня"

одного ответа, и если среди первых пяти оказался верный, то классификация считается верной.

До 2012 года лучшие результаты в этом соревновании показывали методы, количество ошибок у которых достигало 25%. Позже лидерство захватили глубокие нейронные сети. Первое их применение показало результат в 16% ошибок, то есть их количество снизилось практически в 2 раза.

4.1.2. Нейронные сети

Можно сказать, что в 2012 году произошла революция в компьютерном зрении. Она началась с того, что с их помощью была одержана победа в соревновании ImageNet. Далее для всей большего количества задач компьютерного зрения было показано, что нейронные сети работают лучше, чем традиционные подходы.

В предыдущих курсах специализации было разобрано, что из себя представляют нейронные сети. Далее упор будет сделан на сети, которые используются для классификации изображений.

Одно из ключевых отличий этих сетей — наличие свёрточных слоёв. Ранее было разобрано, как свёртка используется для обработки изображений. В случай нейронных сетей происходит то же самое, но она применяется не к картинке, а к выходам из предыдущего слоя. Таким образом, каждый слой состоит из банка фильтров.

Помимо свёрточных слоёв в нейронных сетях для классификации изображений применяется операция под названием пулинг. При этом выбирается из нескольких элементов один, а остальные выбрасываются. В частности, тах-пулинг выбирает элемент с максимальным значением.

Другой трюк, который применялся в 2012 году, — это *dropout*. При его использовании обнуляется часть выходов из предыдущего слоя. Можно считать это методом регуляризации, используемом для того, чтобы сеть не переобучалась. Многие считают *dropout* очень неинтуитивным методом регуляризации, тем не менее, на практике показано, что он очень хорошо работает.

Ещё один метод, применяющийся в соревновании ImageNet, — это дополнение обучающих данных. Чем больше выборка, на которой обучается нейронная сеть, тем лучше она впоследствии работает. Исходя из этого, кажется логичным пополнять коллекцию различными способами. В частности, в работе 2012 года применялся метод вырезания кусочка картинки, так, чтобы класс не изменялся. Кроме того, вносились цветовые шумы, производилось зеркальное отражение и производились другие изменения, которые не влияли на содержание

картинки. При обучении на этих данных добавались инвариантности сети к таким изменениям.

4.1.3. AlexNet

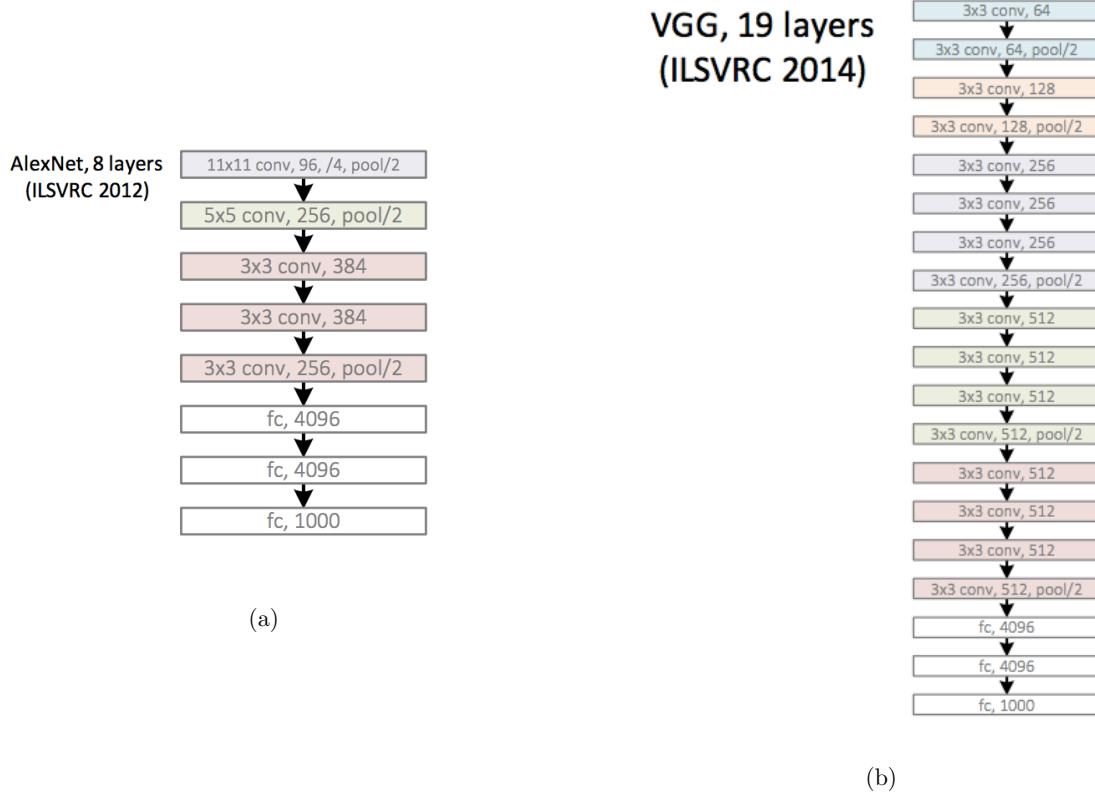


Рис. 4.3: Архитектуры нейронных сетей. (а) — AlexNet, (б) — VGG.

Архитектура знаменитой нейронной сети, победившей в конкурсе в 2012 году, показана на рис. 4.3а. Её разработал и реализовал Алекс Крижевский. Сеть состоит из пяти свёрточных слоёв, двух полносвязных слоёв и выходного слоя с 1000 выходами. Количество выходов совпадает с числом классов в задаче. Таким образом, выход с максимальным значением показывает класс, к которому принадлежит изображение. Данная сеть допустила 16.5% ошибок. Впоследствии возникло много других архитектур, улучшивших этот результат.

4.1.4. Ансамбль сетей

Первое заметное улучшение качества классификации было получено не с помощью изменения архитектуры, а при использовании ансамблей. Идея заключалась в том, чтобы обучить несколько нейронных сетей, каждую картинку классифицировать всеми сетями, а результат усреднить. Такая несложная операция позволила уменьшить долю ошибок с 16.5% до 11.7%.

4.1.5. VGG

В 2014 году группа исследователей из Оксфорда предложила новую архитектуру нейронной сети. Она состояла из 19 слоёв, большая часть из них — свёрточные (все они размера 3×3). Данная сеть позволила улучшить результат классификации и существенно снизить ошибку до 7.3%.

4.1.6. GoogleNet

Ещё одна архитектура — это GoogleNet. Эта нейронная сеть состоит из компонент, показанных на рисунке 4.4. Основная идея состоит в том, что на каждом слое используется не одна свёртка, а несколько, причём

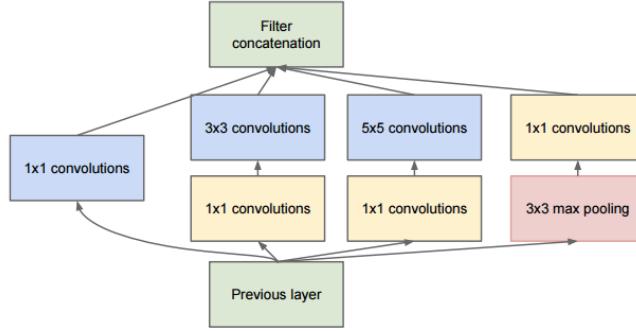


Рис. 4.4: Компоненты свёрток нейронной сети GoogleNet.

разного размера. Это помогает реагировать на сигналы разного масштаба и улучшает качество работы (доля ошибок 6.7%). Полный вид этой сети показан на рисунке 4.5а.

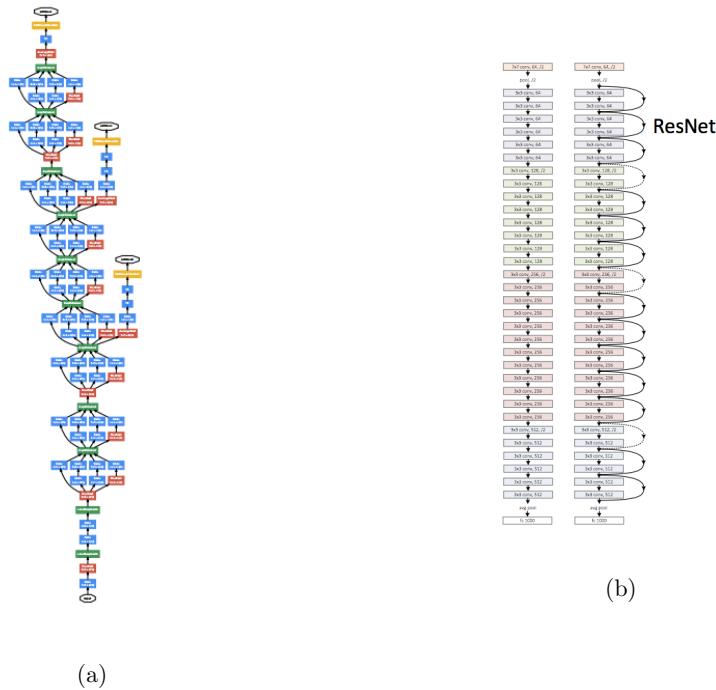


Рис. 4.5: Архитектуры нейронных сетей. (а) — GoogleNet, (б) — ResNet.

4.1.7. ResNet

На рисунке 4.6 показано, как менялись результаты соревнования ImageNet из года в год. Последняя архитектура состоит из 152 слоёв, это Residual Neural Network. С её помощью доля ошибок была уменьшена до 3.57%. Ключевым элементом данной архитектуры является связь, которая пропускает несколько слоёв, передавая результат предыдущего слоя. Такое изменение позволило полностью отказаться от таких методов регуляризации, как DropOut.

Итак, с 2012 года доля ошибок классификации изображений уменьшилась почти в 4 раза, с 16.5% до 3.57%. Это большой скачок, который был осуществлён благодаря развитию нейронных сетей и глубокого обучения. Помимо задачи классификации, нейронные сети применяются для решения практически всех задач компьютерного зрения.

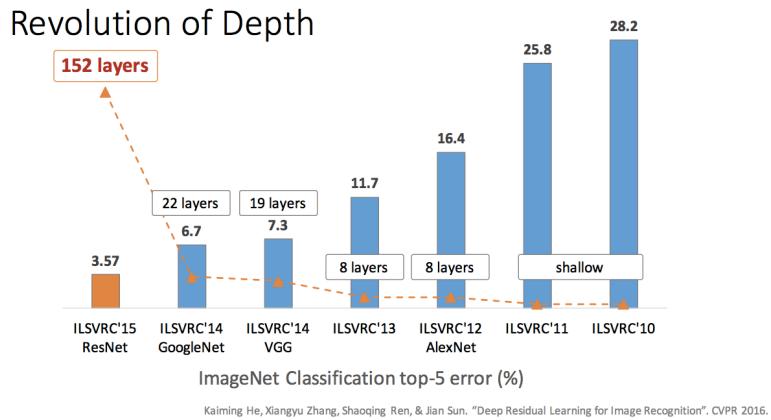


Рис. 4.6: Эволюция ошибки классификации в соревновании ImageNet

4.2. Задача классификации изображений на практике

Далее речь пойдёт о том, как решать задачу классификации изображений на практике.

Прежде всего необходимо собрать базу изображений, для которых известно, к какому классу они принадлежат. Затем нужно определиться с алгоритмом классификации и предобработать картинки. Далее будет разобрано подробнее, как это делать. Последний шаг — это обучение нейронной сети.

4.2.1. Библиотеки для работы с нейросетями

Существует множество библиотек для работы с нейронными сетями. Далее перечислена лишь часть их них:

torch7 написана на C++, всё взаимодействие с клиентским кодом происходит через Lua. Существует достаточно давно, поэтому имеется много реализаций разных задач;

tensorflow — относительно новая, очень удобная библиотека. Минус заключается в том, что из-за новизны в ней часто нет реализаций, которые присутствуют в других библиотеках. С другой стороны, это обычно просто сделать;

theano существует достаточно давно, очень популярна в академических кругах (учёные любят использовать её для иллюстрации статей и быстрых экспериментов). Минус этой библиотеки — её низкоуронвность (требуется много сил и опыта, чтобы понять, как с её помощью работать с сетями), в библиотеке реализованы тензорные вычисления, а не нейронные сети. По этой причине, как правило, работа происходит не с theano, а с обёртками;

keras — обёртка для theano;

lasagne — обёртка для theano;

caffe — одна из первых библиотек для работы со свёрточными нейронными сетями, реализована на C++. Плюс библиотеки: она часто используется для production-задач (потому что написана на C++, а также часть применения нейронных сетей хорошо отлита).

При выборе библиотеки стоит учитывать разнообразные факторы. Например, любовь разработчиков к тому или иному языку программирования (кто-то предпочитает python, кто-то — lua). Также стоит посмотреть, кто уже решал похожую задачу до этого, и в каком фреймворке, и взять эту реализацию за основу. Этот фактор часто является определяющим при выборе библиотеки: если что-то уже реализовано на torch7, зачем использовать что-то другое.

В целом переключаться между этими библиотеками не так сложно, их идеологии достаточно похожи. Поэтому, разобравшись с одной библиотекой, можно воспользоваться и другой. Также существуют конвертеры моделей между фреймворками, что облегчает переход.

4.2.2. Tensorflow

Плюсы библиотеки tensorflow — это богатая документация, большое количество тьюториалов, достаточно простой python api, ядро реализовано на C++.

Часто решение задачи классификации делится на две части. Первая — это подбор модели, обучение классификаторов. Вторая часть — внедрение классификаторов в production. На этом этапе нужна только та часть библиотеки, которая связана с исполнением модели. В некоторых библиотеках (в частности, tensorflow) эту часть легко отделить или можно использовать всю библиотеку, и это никак не повлияет на производительность, а в других — могут возникать проблемы.

4.2.3. Зоопарк моделей

При решении задачи в первую очередь необходимо посмотреть, не решал ли её кто-то до этого, и нет ли готового решения. Для библиотеки caffe существует единый репозиторий моделей (зоопарк моделей, <https://github.com/BVLC/caffe/wiki/Model-Zoo>), можно попробовать искать решение в нём. В частности, там выложены несколько моделей, обученных на базе ImageNet, причём с лучшими результатами на момент обучения. Таким образом, возможно, ничего не нужно придумывать, задача уже кем-то решена. Тогда достаточно взять готовую модель и использовать её на практике.

4.2.4. VGG

Одна из самых часто используемых моделей — это модель VGG, разработанная оксфордской группой исследователей (рис. 4.3b). Её устройство обсуждалось ранее. Реализация этой архитектуры представлена в зоопарке моделей.

Плюсы этой архитектуры заключаются в том, что у неё очень простая структура, она наиболее кроссплатформенна между библиотеками, и, скорее всего, поддерживается любой библиотекой нейронных сетей, которая используется на практике. Кроме того, она показывает достаточно хороший результат на базе ImageNet (не самый лучший, но входит в тройку лидеров).

Существует готовая VGG модель для caffe, кроме того, есть конвертер из этой библиотеки в любую другую.

4.2.5. Дообучение

Итак, стоит задача выполнить классификацию изображений на категории, которых изначально нет в ImageNet. Коллекция уже собрана, теперь есть несколько вариантов развития событий. Можно обучить на этой коллекции нейронную сеть с нуля, как это делалось для задачи ImageNet. Однако сделать это не так просто. Во-первых, коллекция должна быть достаточно большой, а это часто невозможно (разметить миллион картинок — трудоёмкое занятие). Другой вариант — воспользоваться обученной моделью, например, VGG из зоопарка моделей, и дообучить её на собранных данных.

Дообучение можно производить несколькими способами. Во-первых, можно убрать самый последний слой, в котором осуществлялась классификация в ImageNet, и заменить его на новый слой с необходимым количеством классов. Дообучение будет производиться только между этим слоем и последним полносвязным. Плюсы такого подхода заключаются в том, что его достаточно просто осуществить, для этого нужно не так много изображений, не требуется больших мощностей (достаточно среднего ноутбука), сеть впоследствии показывает хорошие результаты. Пример дообучения в библиотеке tensorflow приведён по ссылке (https://www.tensorflow.org/versions/r0.9/how_tos/image_retraining/index.html#distortions).

Другой вариант также предполагает использование готовой, уже обученной на базе ImageNet модели. В ней необходимо заменить последний слой, а затем дообучить все слои, а не только переход от последнего полносвязного слоя к ответу. Для этого требуется больше вычислительных мощностей, но так как сеть уже предобучена, это займёт меньше времени, чем обучение сети с нуля.

Выбор между двумя описанными вариантами зависит от задачи, размера имеющейся базы и желаемого качества классификации. Стоит повторить, что дообучение последнего перехода очень часто даёт хорошие результаты несмотря на то, что это занимает гораздо меньше времени, чем обучение всей сети.

Поиск изображений

Поиск изображений — это ещё одна задача, в которой используются предобученные нейронные сети. Для демонстрации возможного решения этой задачи будет использоваться модель AlexNet (рис. ??), при этом считается, что один из последних слоёв описывает изображение. Это может быть пятый свёрточный слой,

шестой или седьмой полно связанный. При поиске изображения оно будет описываться вектором признаков, взятым из одного из этих слоёв.



Рис. 4.7: Примеры результата поиска похожих изображений. Слева — запрос, остальные картинки — ответы.

Возникает вопрос, как сравнить картинки, чтобы определить, похожи они или нет. Самый простой способ — использовать для этого евклидово расстояние. На рисунке 4.7 показан результат поиска похожих изображений для различных вариантов выбора слоя, описывающего картинку. Видно, что приемлемые результаты получаются, если выбирать в качестве описания седьмой слой: фотографии той же двери расположены на втором и третьем месте. Данный пример является довольно сложным, поэтому результат выходит не очень хорошим. Подробные результаты можно найти по ссылке (<http://sites.skoltech.ru/compvision/projects/neuralcodes/>). Там представлены подробные соображения о том, какой слой выбирать для описания, как сравнивать вектора признаков, какие комбинации работают лучше в конкретных ситуациях.

4.2.6. Нейронная сеть — хорошее представление

Хочется отметить следующее. Выходы последних слоёв нейронной сети использовались для дообучения классификаторов в разнообразных задачах. Эти же выходы применялись для поиска изображений. Кроме того, нейронная сеть обучалась на коллекции ImageNet, а использовалась для совершенно других наборов изображений. Как показывает практика, эти подходы работают хорошо, даже если модель используется для работы с изображениями абсолютно другой природы (например, с компьютерной графикой, в коллекции ImageNet её нет). Все эти факты говорят о том, что нейронные сети, обученные на большой базе разнообразных изображений, достаточно хорошо описывают картинки. Это фундаментальный результат, показывающий, что данное представление можно использовать в совершенно разных задачах.

4.3. Распознавание лиц

Задачу распознавания лиц можно разделить на две подзадачи. Первая — верификация лиц, требуется по двум фотографиям определить, изображён на них один и тот же человек или нет. Вторая подзадача — распознавание, по фотографии необходимо определить, присутствует ли этот человек в базе (например, базе фотографий злоумышленников), и найти его в ней. Задача распознавания вытекает из задачи верификации, и если научиться хорошо решать первую, то можно решить и вторую. В любом случае, необходимо иметь хорошее представление лиц в сырье виде. Ранее рассказывалось о том, что с помощью нейронных сетей можно получить хорошее представление изображений, остаётся применить это знание.

4.3.1. Базы лиц

Для решения любой задачи машинного обучения (в частности, компьютерного зрения) требуется размеченная база. В мире существует несколько публичных баз лиц. В последнее время чаще всего алгоритмы сравнивают между собой по базе Labeled faces in the wild. Она содержит 13 тыс. фотографий, взятых из интернета. База Megaface содержит 5 млн фотографий, на которых запечатлены 672 тыс. людей. Во второй базе хранится гораздо больше изображений, и они более приближены к реальности. Однако она появилась недавно, не все алгоритмы были на ней протестированы, поэтому для сравнения качества работы всё ещё используется первая база.

4.3.2. Пары лиц



(a)

(b)

Рис. 4.8

Чтобы оценить сложность задачи верификации, полезно рассмотреть пример. На рисунке 4.8 представлены две пары изображений, на одной из них разные люди, а на другой — один и тот же человек. На самом деле, не все люди могут сразу определить, на какой паре изображён один и тот же человек. Плохое качество изображений связано не с желанием усложнить задачу, оно такое изначально, потому что лица людей, вырезанные с фотографий из интернета, получаются маленькими. Фотографии такого же качества даются на вход нейронной сети.

4.3.3. Фреймворк распознавания

Почти все алгоритмы классификации и распознавания лиц можно разделить на несколько фаз. Сначала необходимо найти на фотографии лицо и вырезать. Затем его нужно развернуть таким образом, чтобы оно имело примерно одно положение на всех фотографиях. Это упрощает работу алгоритмов классификации. Из полученной картинки генерируется цифровое представление лица. Последний шаг — сравнение полученных лиц. Далее каждый из этих этапов будет рассмотрен подробнее.

Детектор

Ранее упоминалось, что существует много методов детекции лица. Один из первых успешных использует каскады Хаара, сейчас это не самый лучший и успешный метод. В данный момент для решения этой задачи используются нейронные сети. Кроме того, достаточно хорошим компромиссом между качеством, скоростью и удобством использования является следующий метод. Из изображения извлекаются дескрипторы hog, а затем по ним алгоритм машинного обучения svm определяет, лицо это или нет.

Существуют методы выделения особенностей лица, таких как кончик носа, уголки рта, края глаз и т.д. Зная эти особые точки, можно тем или иным способом развернуть лицо.

Выравнивание

Существует несколько способов выравнивания лица. Можно развернуть изображение в плоскости, выполнив двумерное преобразование. Другой способ — попытаться построить трёхмерную форму лица, и сделать так, будто человек смотрит прямо в камеру.

До появления нейронных сетей качество выравнивания было одним из ключевых параметров алгоритма распознавания лиц. В соревнованиях побеждали методы, создающие качественное 3D-представление, разворачивающие лицо, дорисовывающие вторую половину лица, если на фото изображён профиль. Однако нейронные сети могут хорошо работать даже на не до конца выравненных лицах, поэтому требования к этому этапу работы алгоритма снизились. Более того, простые методы (например, поворот в плоскости) даже выигрывают, потому что сложные методы выравнивания вносят шум в изображение.

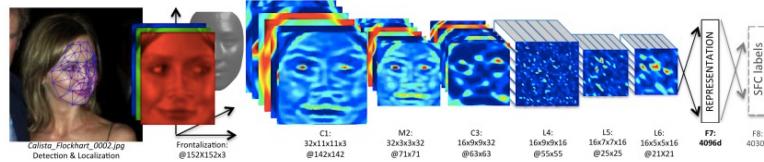


Рис. 4.9: Архитектура нейронной сети, использующаяся для классификации лиц (классы — люди).

Обучение представления

После того как лицо найдено и выравнено, можно перейти к работе непосредственно с картинками. Для этого применяется уже не раз упоминавшаяся схема классификации изображений с использованием глубоких нейронных сетей. На рисунке 4.9 показана похожая на AlexNet архитектура, которая применяется для классификации лиц в статье. Эта нейронная сеть состоит из нескольких свёрточных и полно связанных слоёв, она обучалась на базе, состоящей из нескольких миллионов фотографий, в роли классов выступают различные люди. После обучения сети выходной слой использовался в качестве описания картинки, с помощью которого решалась задача верификации на базе Labeled faces in the wild.

Возникает вопрос, как сравнивать между собой представления изображений. До появления нейронных сетей это было важным компонентом всей цепочки метода, и были придуманы различные способы производить сравнение. В случае нейронных сетей хорошо работает обычное евклидово расстояние: если оно больше заданного порога, то люди разные. Сложные методы либо вообще не дают никакого улучшения в этом случае, либо это улучшение совсем небольшое.

В статье, использующей описанную выше архитектуру, была достигнута точность 97.35%. Это первая работа на базе нейронных сетей, которая значительно улучшила результат (ошибка снизилась в несколько раз, точность используемых ранее методов составляла около 92%).

Ансамбль сетей — deep id

Далее авторы из Гонконга предложили использовать похожий пайплайн и похожую сеть, они добились увеличения точности до 99.47%. Различие заключалось в том, что для этого использовалось 200 нейронных сетей, результаты которых специальным образом объединялись и усреднялись. Стоит отметить, что даже если нейронная сеть небольшая, применять её 200 раз — сложная процедура.

Оксфорд

Ещё один хороший результат был получен группой из Оксфорда, при этом использовалась одна нейронная сеть. Они изменили протокол обучения, база лиц была относительно небольшой (2 млн картинок). При этом было получено качество 98.95%. Впоследствии результат был улучшен до 99.13%, для этого было изменено обучение сравнения дескрипторов.

End-to-end

Как уже было сказано ранее, фреймворк распознавания состоит из детектирования лица, его выравнивания и обучения нейронной сети. Сотрудники компании Google решили, что все эти шаги могут выполняться внутри сети. Они использовали базу из 200 млн изображений и без предобработки обучили на ней сеть. Полученное качество составляло 98.87%. Кажется, что большой размер базы должен способствовать тому, чтобы нейронная сеть сама научилась выравнивать изображения. Однако даже в этом случае предварительное выравнивание сильно помогает, с его использованием качество достигло 99.63%.

4.3.4. Результаты

В таблице 4.1 суммированы результаты верификации лиц, полученные с использованием различных методов. Важно учитывать не только качество работы метода, но и размер базы, на которой производилось обучение: чем больше база, тем сложнее её собрать. Например, поражает воображение точность, полученная при обучении на базе из 200 млн изображений, но повторить его достаточно сложно. С другой стороны, группа из Оксфорда показала очень достойный результат, используя относительно небольшую базу.

№	Метод	Число изображений, млн	Количество сетей	Точность, %
1	Fisher Vector Faces	-	-	93.10
2	DeepFace	4	3	97.35
3	Fusion	500	5	98.37
4	DeepID-2,3		200	99.47
5	FaceNet	200	1	98.87
6	FaceNet + Alignment	200	1	99.63
7	Ours	2.6	1	98.95

Таблица 4.1: Результаты верификации лиц различными методами

Таким образом можно считать, что на базе Labeled faces in the wild, состоящей из 13 тыс. изображений, задача распознавания решена достаточно хорошо, с точностью почти 100%.

4.3.5. Поиск похожих лиц

Ещё один пример использования представления, полученного при использовании классификаторов, — это поиск похожих лиц. Так же как в задаче поиска похожих изображений, для этого можно использовать несколько последних слоёв нейронной сети. В случае, если в базе уже есть данный человек, будет найден он, иначе — просто похожие люди. В данный момент существует несколько сервисов, работающих по этому принципу.

4.3.6. t-sne

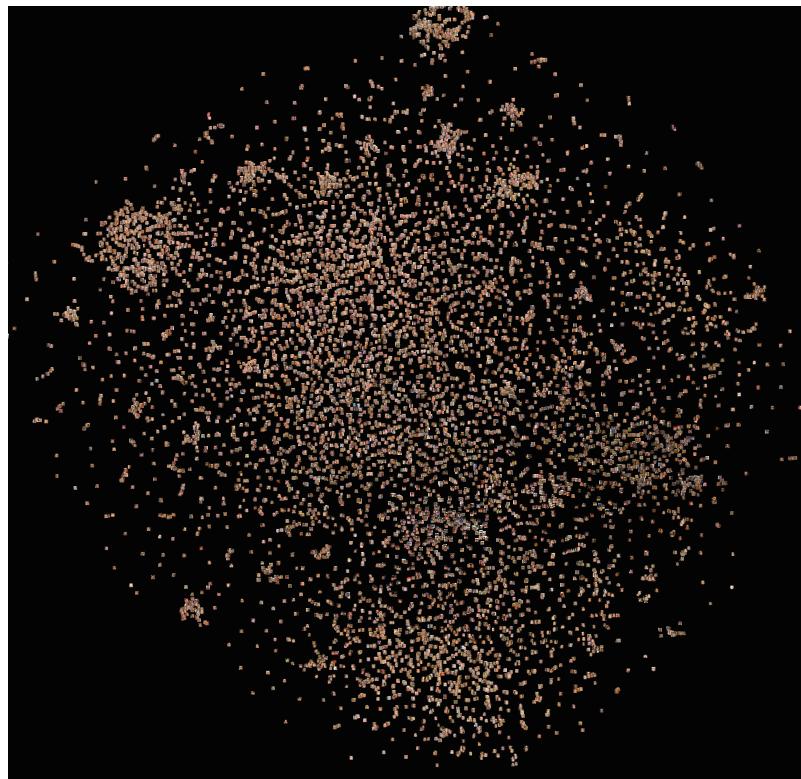


Рис. 4.10: Визуализация на плоскости признаковых описаний лиц, полученных с помощью нейронных сетей.

Чтобы лучше понять, как устроены представления лиц с помощью нейронных сетей, можно визуализировать облако векторов на плоскости (4.10). Для этого можно использовать метод t-SNE, который располагает вектора на плоскости таким образом, чтобы они были тем ближе, чем ближе в исходном пространстве. На рисунке есть несколько сгущений, можно их приблизить и посмотреть, что они из себя представляют (рис.

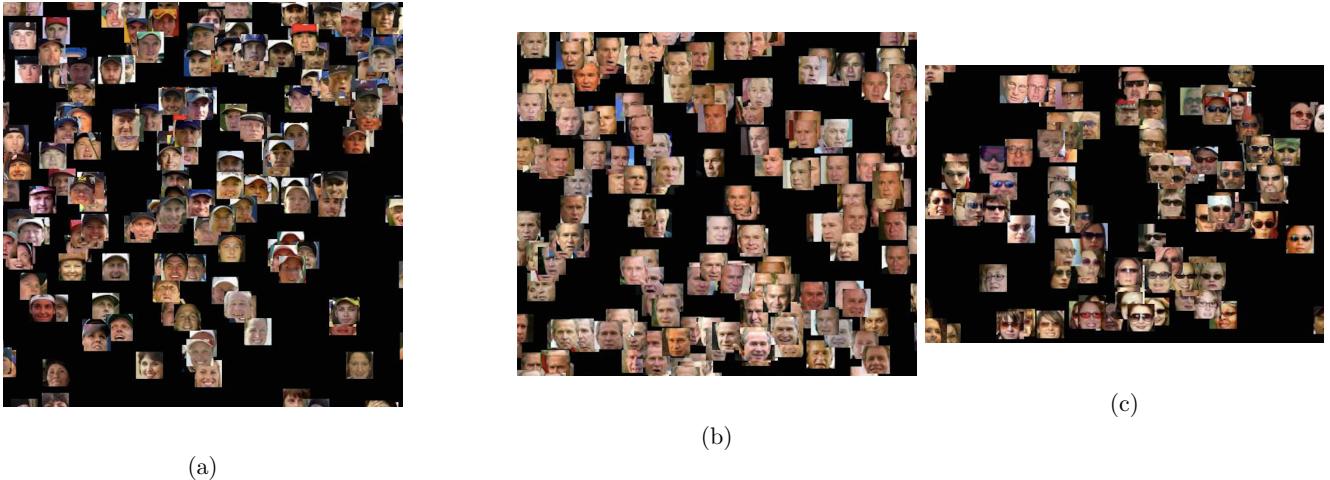


Рис. 4.11: Сгущения из рисунка 4.10. (а) — люди в кепках, (б) — Джордж Буш, (с) — люди в очках

4.11). Интересно, что данные сгущения никак не помогают классифицировать людей. Например, тёмные очки (рис. 4.11c) могут присутствовать у разных людей. Тем не менее, нейронная сеть обратила на это внимание, и считает людей в солнцезащитных очках похожими. Это означает, что полученные признаки неидеальны, и, возможно, произошло переобучение.

4.3.7. Резюме

За последние несколько лет в задаче распознавания лиц произошёл большой прогресс, многократно увеличилось качество работы алгоритмов на одной из баз. На базе Labeled faces in the wild получены результаты с точностью почти 100%, это означает, данная коллекция исчерпала себя. Цель на ближайшие годы для исследователей в этой области — добиться хорошего качества на базе Megaface, содержащей большее количество разнообразных изображений. Если эта цель будет достигнута, это означает, что алгоритмы компьютерного зрения будут хорошо работать на огромных коллекциях (социальные сети, изображения с уличных камер и камер в метро).

Урок 5

Практические задачи компьютерного зрения

5.1. Детекция объектов

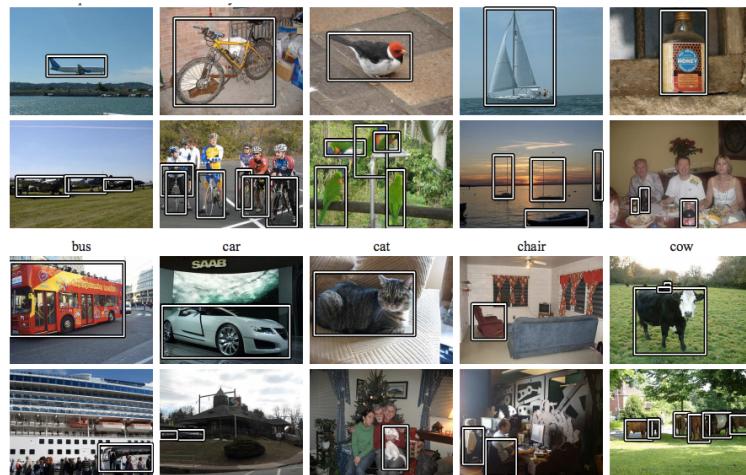


Рис. 5.1: Детекция объектов

Далее будет разобрано несколько практических задач. Первая из них — это детекция объектов. До этого речь шла о задаче классификации, когда по целой картинке требовалось определить, к какому классу она принадлежит. Задача детекции объектов подразумевает, что необходимо не только назвать класс изображения, но и найти на нём место, где расположен объект.

5.1.1. Скользящее окно

Один из традиционных подходов — это скользящее окно. Он уже обсуждался в контексте задачи фильтрации изображений и в данном случае не сильно отличается. Окно перемещается по картинке, к каждому такому окну можно применить классификационную нейронную сеть, обученную на тех классах, объекты которых требуется детектировать. В случае, если сеть обнаружила присутствие объекта в окне, оно помечается рамкой и классом.

R-CNN

У этого подхода есть недостаток. Чтобы пройти скользящим окном по изображению, ещё и на разных масштабах, нужно многократно применять классификационную нейронную сеть, а значит, это очень медленный процесс. Чтобы его ускорить, были придуманы разнообразные эвристики. Один из очевидных способов —

применять не тяжёлую классификационную сеть, а некий классификатор, отбрасывающий окна, в которых явно нет искомых объектов. Этот подход называется R-CNN (<https://github.com/rbgirshick/rcnn>), в нём с помощью метода Selective Search отбираются окна, а затем на них запускается классификационная нейронная сеть. R-CNN — это достаточно простой метод, он до сих пор используется как baseline при детекции объектов.

Faster R-CNN

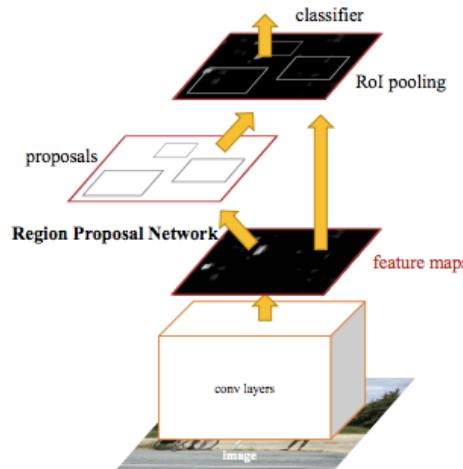


Рис. 5.2: Принцип работы метода Faster R-CNN

Несмотря на то что предыдущий метод достаточно хорошо себя показал, существует пространство для его улучшения. Та же группа авторов предложила метод R-CNN (рис. 5.2), где для предложения гипотез об окнах не используется внешний классификатор. В данном случае нейронная сеть выдвигает гипотезы об окнах и классифицирует. Вторая инновация заключается в том, что выдвигается не только гипотеза о наличии объекта в окне, но и уточняется его положение внутри. Идея оказалась успешной, и в данный момент этот метод является лучший по предсказанию объектов.

YOLO

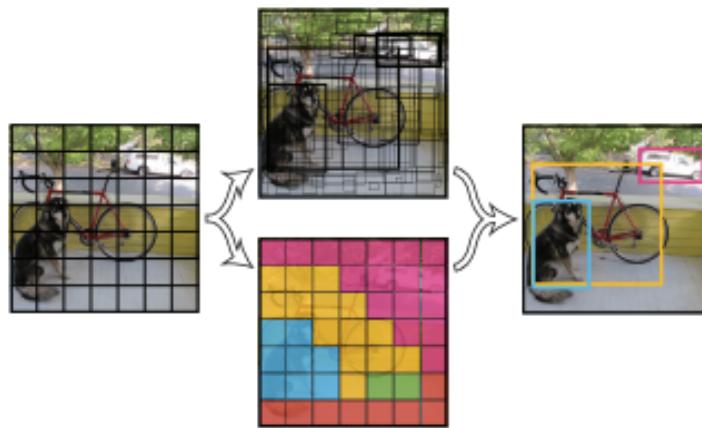


Рис. 5.3: Принцип работы метода YOLO

Аналогичная идея использования одной нейронной сети для предсказания класса объекта и положения ограничивающего прямоугольника использовалась при создании метода YOLO (рис. 5.3). Этот подход яв-

ляется ещё более простым. Картинка делится на несколько ячеек, в каждой из которых классификатор применяется отдельно. После этого строится предсказание о классе объекта и положении ограничивающего прямоугольника. Метод Yolo работает очень быстро, на рисунке 5.4 представлены результаты сравнения качества и скорости различных подходов. Faster R-CNN показывает лучший результат по качеству, скорость работы при этом около 7 кадров в секунду. В то же время, метод YOLO работает с качеством на 10% хуже, зато скорость работы достигает 155 кадров в секунду, что позволяет использовать его для детекции объектов в режиме реального времени.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
<hr/>			
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

Рис. 5.4: Результаты сравнения различных методов детекции объектов по скорости и качеству

5.1.2. Генеративные сети

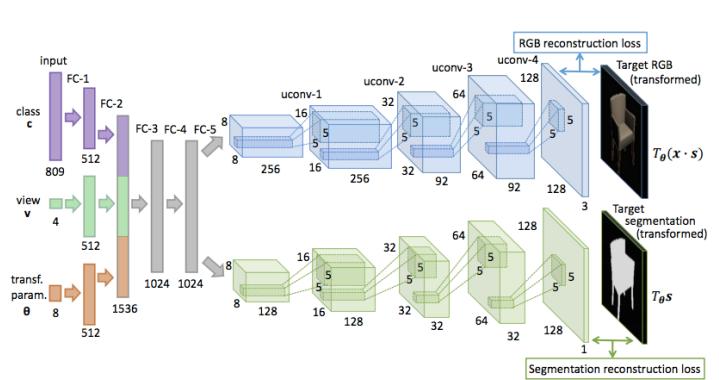


Рис. 5.5: Генеративная нейронная сеть

До этого обсуждались нейронные сети, которые по картинке выдают какие-то её параметры: либо класс объекта, либо координаты ограничивающего прямоугольника. Генеративные нейронные сети — это другой класс сетей, которые из векторов признаков или некоторых параметров умеют генерировать изображение. В одной работе демонстрировался интересный пример такой сети (рис. 5.5). На вход подаётся класс объекта, его положение в пространстве и то, с какой стороны на него смотрят, а нейронная сеть генерирует соответствующий объект. В качестве таких объектов в статье использовались стулья. На рисунке 5.6 в крайних столбцах находятся стулья из коллекций, а между ними — сгенерированные данной сетью. Видно, что промежуточные результаты похожи на стулья, и с помощью простого усреднения добиться такого результата не получилось бы. Это означает, что нейронная сеть в процессе обучения "осознала" форму стула, и использовала это для генерации новых.



Рис. 5.6: Крайние столбцы — реальные стулья, остальные — сгенерированные нейронной сетью

5.1.3. Семантическая сегментация

Далее будет под示范ировано, как можно использовать описанные нейронные сети для различных приложений. Изначально рассматривалась задача классификации, в которой необходимо что-то сказать о картинке целиком. В этой части была описана задача детекции объектов, в которой требуется найти на изображении объект, и что-то сказать о нём. Можно пойти дальше и для каждого пиксельного изображения на картинке определить, к какому классу оно принадлежит. На рис. 5.7 показан пример решения задачи сегментации, каждый класс на нём закодирован цветом. Это фотография улицы европейского города. На ней изображены объекты дорожной инфраструктуры: дорога, тротуар, велосипедисты, трамвай, люди, дорожные знаки. Задача состояла в том, чтобы найти все подобные классы на изображении.



Рис. 5.7: Пример решения задачи сегментации

Fully convolutional networks

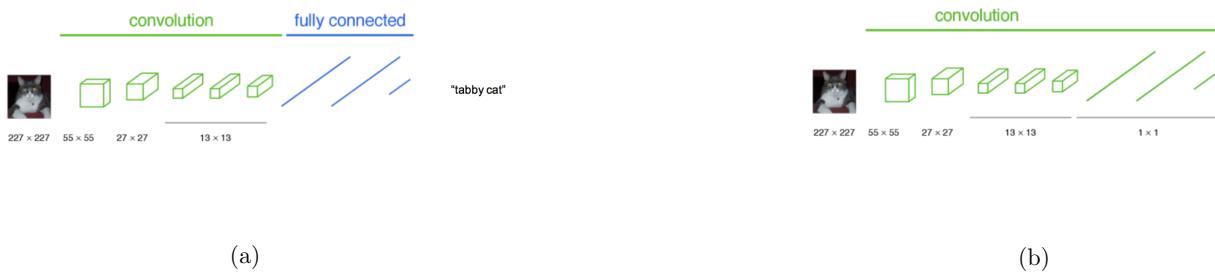


Рис. 5.8: (а) — свёрточная нейронная сеть, (б) — полностью свёрточная нейронная сеть

Существует несколько методов, которые решают описанную выше задачу. Один из них — это полностью свёрточные сети (fully convolutional networks). Ранее, когда речь шла о нейронных сетях, они состояли из нескольких свёрточных слоёв, а после них шло ещё несколько полно связанных слоёв (рис. 5.8а). На самом деле, нет никаких причин, по которым последние слои нельзя заменить на свёрточные (5.8б). Прелесть свёрточных слоёв заключается в том, что размер входной картинки неважен, а картинка на выходе будет пропорциональна исходной.



Рис. 5.9: Полностью свёрточная нейронная сеть

Итак, при использовании нейронной сети, состоящей только из свёрточных слоёв, результат на выходе пропорционален входному изображению. Например, если использовать архитектуру, похожую на AlexNet, но последние полно связанные слои заменить на свёрточные, то выход будет в 32 раза меньше, чем вход (рис. 5.9а). Если на вход нейронной сети подавать большую картинку, то выход будет содержать достаточно много информации. После этого к нему можно применить простые алгоритмы повышения размерности и получить изображение, разрешение которого равно разрешению входа (рис. 5.9б).

После того как появилась такая нейронная сеть, ничто не мешает в явном виде решать задачу сегментации: на вход подаётся изображение, на выходе получается изображение с таким же разрешением, но исходные пиксели заменены на номера классов, к которым они принадлежат. Если имеется обучающая выборка, то нейронную сеть можно обучать в явном виде. Проблема этого подхода заключается в том, что выход нейронной сети намного меньше по размеру, чем необходимое разрешение, а увеличение размерности в 32 раза может сильно портить качество.

В качестве решения авторы подхода предлагают использовать выходы не только последнего слоя, но и промежуточные: они меньше входа не в 32 раза, а в 16 и 8 раз. Агрегируя эти выходы в ответ, можно получить изображение хорошего качества. В некотором смысле такая нейронная сеть эмулирует применение сети на разных масштабах.

SEGNET — это другой метод решения задачи семантической сегментации. Авторы этого подхода использовали полно связную архитектуру, поникающую размерность, и развернули её. Вопрос состоит в том, как повышать размерность. Для понижения используется операция тах-пулинг (из окрестности 2×2 выбирается максимальное значение, которое используется далее), каждое её применение понижает размерность в 2 раза. Для повышения размерности существует несколько методов. Один из них заключается в том, чтобы запоминать индексы максимумов, взятых при проведении тах-пулинга, хранить их до слоёв с обратным тах-пулингом, и на этом слое выбирать значения с теми же индексами. Такая конструкция хорошо работает и показывает результаты, превосходящие полностью свёрточные нейронные сети. Существуют эвристики

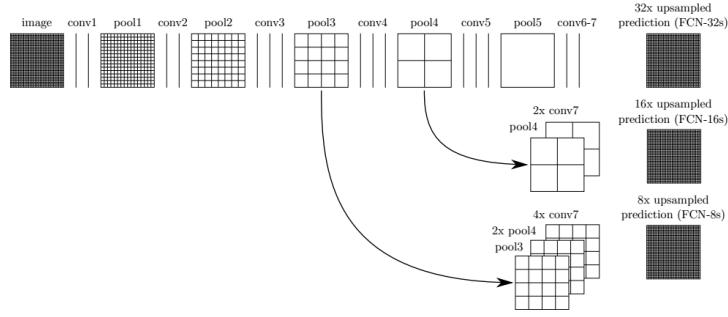


Рис. 5.10: Архитектура нейронной сети, решающей задачу семантической сегментации и использующей не только последний слой (https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf)

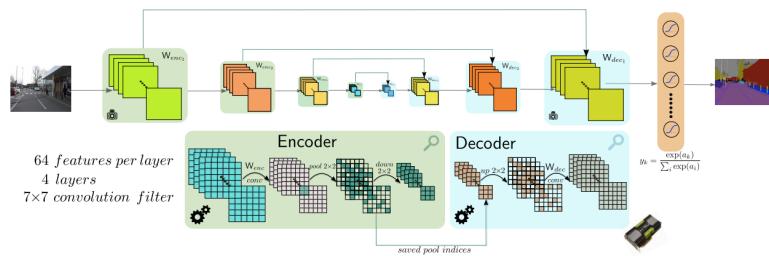


Рис. 5.11: Архитектура SEGNET

для улучшения этого подхода. В целом кажется, что такие нейронные сети представляют интерес не только применительно к задаче семантической сегментации.

5.2. Стилизация изображений

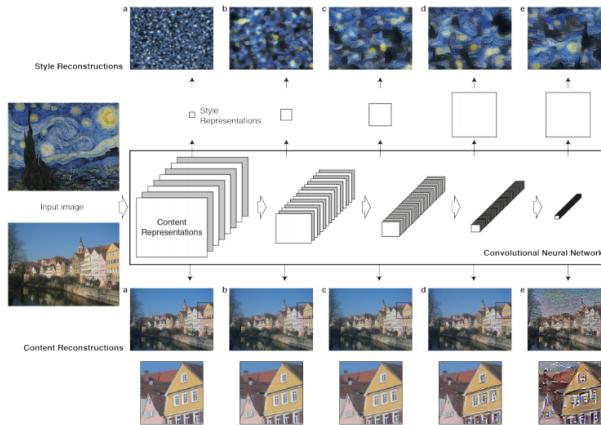


Рис. 5.12: Архитектура нейронной сети, выполняющей стилизацию изображений <http://arxiv.org/pdf/1508.06576.pdf>

Задача стилизации изображений заключается в том, что необходимо перенести стиль с картины художника на фотографию. Ученые из университета Тюбингена предложили алгоритм генерации таких картинок. Ключевую роль в нём играет предобученная нейронная сеть (использовалась архитектура VGG, как и во многих задачах компьютерного зрения). Фактически она оценивает, насколько сгенерированная картинка похожа на картину художника по стилю, а на фотографию — по содержанию. В данном случае веса нейронной сети оставались фиксированными, менялось изображение на входе.

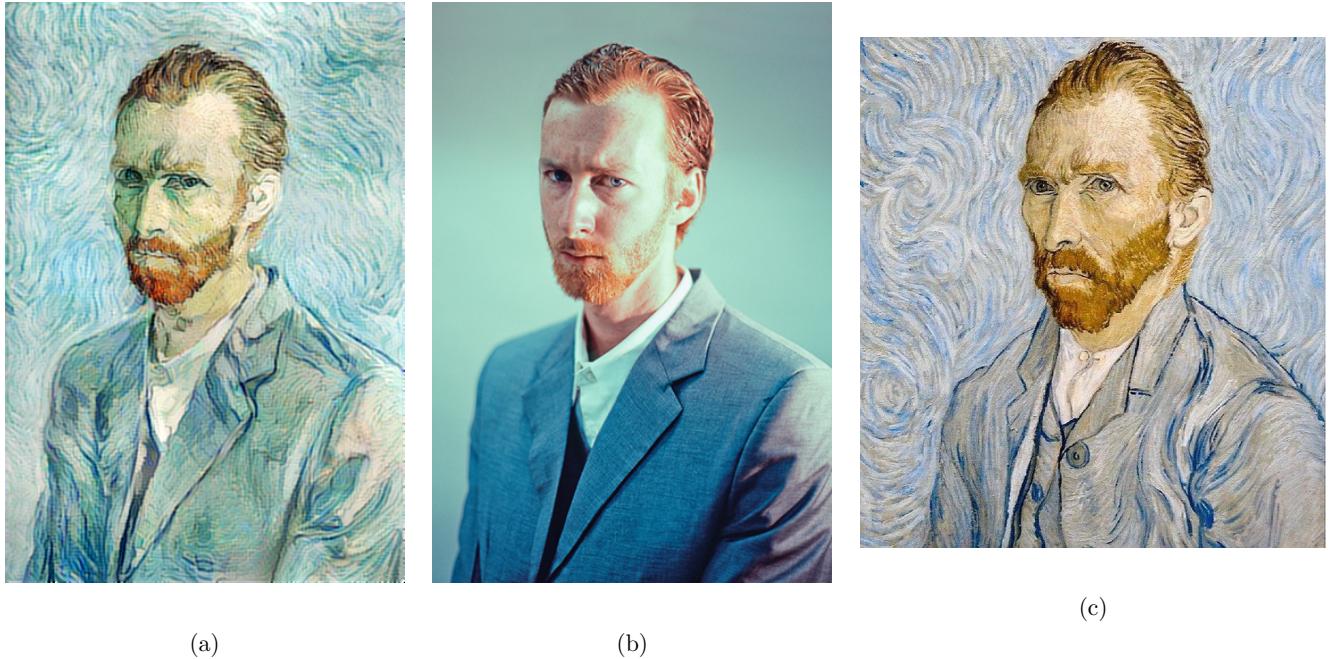


Рис. 5.13: (а) — фотография, стилизованная под картину Ван Гога, (б) — исходная фотография, (с) — картина Ван Гога

Проблема этого подхода заключалась в производительности. На генерацию одного изображения уходили десятки секунд, а иногда и минуты, в зависимости от разрешения исходной картинки.

Многим кажется, что картина на рисунке 5.13а написана Ван Гогом. На самом деле это фотография рыжего мужчины (рис. 5.13б), стилизованная под "Автопортрет" Ван Гога (рис. 5.13с).

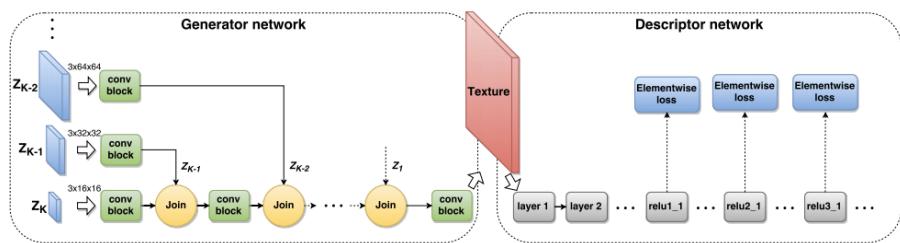


Рис. 5.14: Архитектура нейронной сети, ускоряющей стилизацию изображений (https://github.com/DmitryUlyanov/texture_nets)

Группа учёных из Сколтеха предложила способ ускорения данного алгоритма (рис. 5.14). Для генерации изображения они использовали полношёрточные нейронные сети, описанные в части про семантическую сегментацию. Для оценки, насколько изображение похоже по стилю на картину, а по смыслу на фотографию, используется предобученная сеть VGG. Данный метод работает быстрее, потому что в процессе обучения создаётся такая полношёрточная нейронная сеть, которая впоследствии применяется один раз для генерации картинки, в отличие от предыдущего метода, где для создания картинки использовались методы оптимизации.

Если для оценки похожести использовать только стиль, то описанный выше метод подходит для генерации текстур (рис. 5.16). Если же добавить оценку похожести содержания, то можно создавать стилизованные изображения (рис. 5.16).

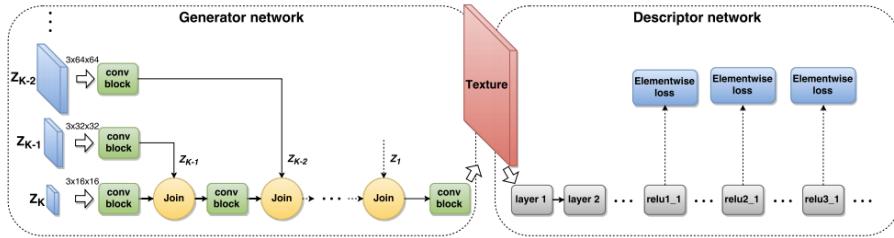


Рис. 5.15: Текстуры, сгенерированные нейронной сетью



Рис. 5.16: Стилизованные изображения, сгенерированные нейронной сетью

5.3. Распознавание китов

Задача распознавания китов была предложена в качестве соревнования на сайте kaggle. Требовалось реализовать распознавание лиц для китов.

В мире осталось очень мало гренландских китов (менее 500 особей). За ними внимательно следят: отслеживают состояние здоровья и пути миграции. Это делается с помощью фотографий с самолётов. Специалисты умеют отличать гренландских китов друг от друга по характерному рисунку из белых наростов на голове. Однако процедура узнавания китов является трудоёмкой, и это умеют делать лишь несколько специалистов. По этой причине создатели конкурса предложили участникам разработать автоматическую процедуру.

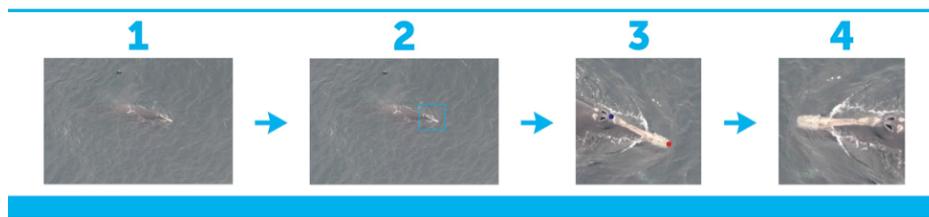


Рис. 5.17: Предобработка изображения в задаче распознавания китов

Данная задача очень сильно похожа на задачу распознавания лиц. Поэтому для её решения можно использовать ту же последовательность действий: детекция головы кита, выравнивание изображения и применение нейронных сетей. Примерно такой подход и был предложен победителями соревнования.

Процесс предобработки фотографии показан на рисунке 5.17. Сначала детектируется голова кита при помощи нейронной сети, которая натренирована отличать голову от фона. Затем другая нейронная сеть находит начало и конец головы. Зная эти две точки, можно развернуть изображение таким образом, чтобы на нём была голова, направленная в нужную сторону. Примеры того, как выглядят предобработанные фотографии с головами китов, показаны на рисунке 5.18. Видно, что по этим изображениям сравнивать китов друг с

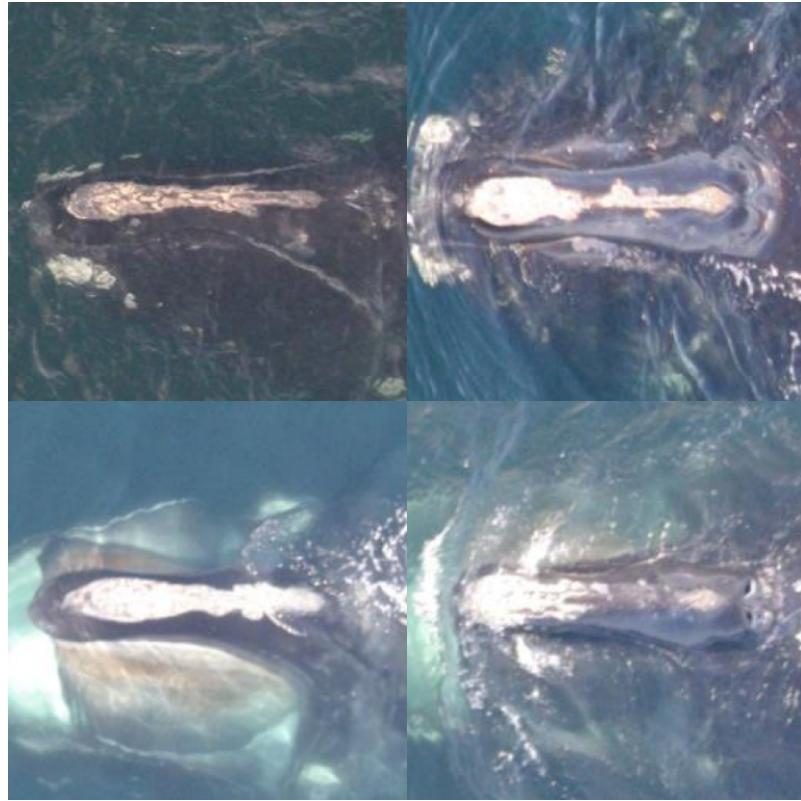


Рис. 5.18: Предобработанные фотографии с головами китов

другом становится гораздо проще.

Далее эти фотографии классифицировались с помощью нейронной сети, архитектура которой изображена на рисунке 5.19. Эта сеть похожа на VGG, но авторы использовали дополнительно несколько трюков, позволивших им улучшить результат.

Один из таких трюков — это увеличение скорости обучения. Обычно при обучении нейронных сетей используется метод уменьшения скорости обучения. Когда график ошибки выходит на плато, скорость обучения уменьшают, и иногда это приводит к тому, что обучение продолжается. Видимо, такой подход не принёс пользы, и авторы решили поступить иначе: увеличить скорость обучения. По графикам (рис. 5.20) видно, что иногда это позволяло продолжить обучение.

Часто соревнования на сайте kaggle сводятся к тому, чтобы попробовать множество различных методов, и найти среди них работающие. Интересно посмотреть, какие методы не сработали у авторов победного решения. Они пробовали вырезать голову кита без обучения, использовать какие-то другие методы обучения. В итоге для части голов пришлось вручную отметить точки начала и конца головы. Этих данных изначально не было.

Также у авторов не сработал метод Spatial transformer networks. Эта нейронная сеть пытается сама обучить преобразование, чтобы качество классификации улучшилось. Видимо, авторы хотели избежать процесса предобработки головы, но это не сработало.

Метод Deep residual networks также не сработал. Ранее упоминалось, что с его помощью хорошо решается задача классификации по базе ImageNet. Авторам это по какой-то причине пользу не принесло.

Наконец, авторы пробовали применять триплеты. В этом методе на вход нейронной сети подаются три картинки: две из них одного класса, третья — другого. Задача нейронной сети сделать так, чтобы изображения из одного класса были друг к другу ближе, чем к изображению из другого класса.

Описанные выше методы достаточно хорошо работают при решении задачи распознавания лиц. Интересно, что они не сработали в задаче распознавания голов китов.

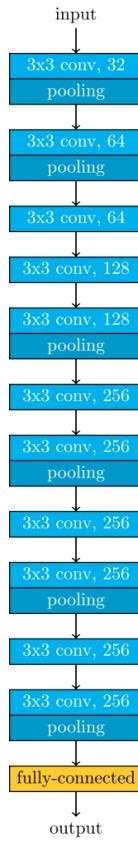


Рис. 5.19: Архитектура нейронной сети для классификации котов

5.4. Сбор больших коллекций изображений

При решении задач компьютерного зрения качество обучающей выборки зачастую оказывается важнее качества самих алгоритмов машинного обучения. Это утверждение можно применить и к анализу данных в целом. Далее речь пойдёт о том, как быстро и просто собрать коллекцию изображений.

5.4.1. Поиск по изображениям

В интернете хранится огромное количество различных изображений, многие из которых снабжены текстовым описанием. Это означает, что можно воспользоваться текстовым поиском по изображениям, чтобы найти картинки необходимого класса. Например, если ввести в поиск по изображениям запрос "машина", то выдача будет состоять из машин.

Группа из Оксфорда, для того чтобы успешно обучить свой классификатор распознавания лиц, собрала собственную базу лиц знаменитостей. Они создали большой список известных людей, затем по нему осуществлялся текстовый поиск по изображениям, и на каждый запрос было получено много фотографий знаменитостей.

Далеко не всегда поисковая система предоставляет качественную выдачу. Вместо того чтобы размечать каждую отдельную картинку, можно размечать запросы. Например, если среди знаменитостей есть однофамильцы, и по запросу, содержащему имя и фамилию, их фотографии выдаются вперемежку, то можно выбросить такой запрос. Кроме того, при поиске фотографий очень известных людей выдаётся много хороших изображений, а для чуть менее известных сначала идут очень хорошие фотографии, а затем начинаются ошибки. Такие запросы также достаточно легко отфильтровать: можно брать только первые несколько картинок, а остальные удалить.

Таким образом группе из Оксфорда удалось собрать базу, состоящую из двух миллионов лиц.

Аналогичным образом поступили исследователи при создании базы достопримечательностей. Идея заключалась в том, чтобы дообучить нейронную сеть на фотографиях достопримечательностей, и определить, насколько это улучшает качество поиска изображений по сравнению с предобученной сетью. Для создания

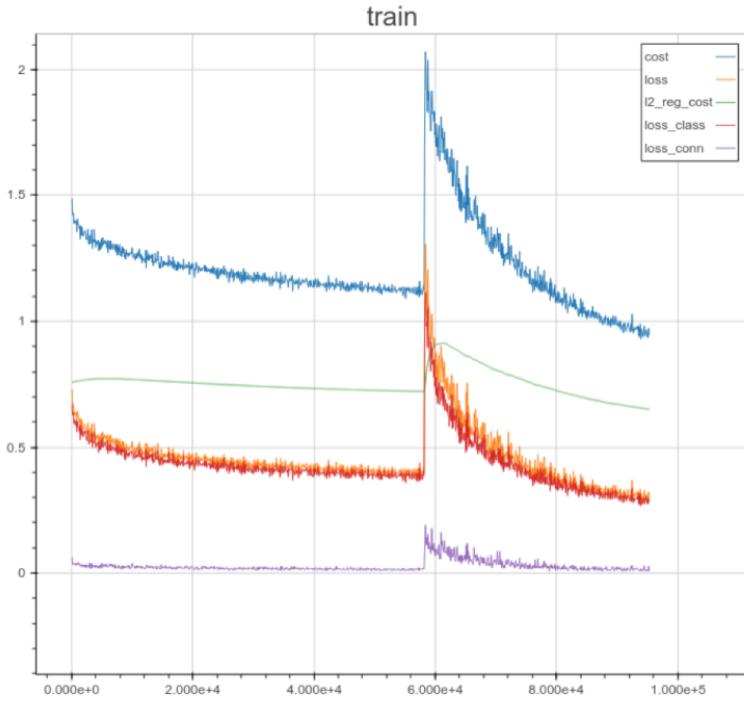


Рис. 5.20: Изменение ошибки нейронной сети при обучении

базы был собран список достопримечательностей, затем оттуда были убраны запросы, содержащие мусор. Например, существуют рестораны, названные в честь достопримечательностей, и иногда поисковые системы выдают изображения интерьеров этих ресторанов. Такие запросы можно выбросить, и достаточно быстро собрать необходимую базу.

5.4.2. Краудсорсинг

Другой способ сбора и разметки изображений — это система краудсорсинга. Существуют системы, позволяющие раздавать людям различные простые задания (например, Amazon Mechanical Turk и Яндекс Толока). С помощью них можно дёшево разметить большое количество изображений, и это удобно.

5.4.3. Итеративное построение базы

Можно использовать и комбинацию: часть изображений получить с помощью поиска по картинкам, а часть — с помощью краудсорсинга. Таким образом можно сформировать итеративную процедуру построения базы.

Для первых экспериментов нужна небольшая, шумная база. По ней обучается классификатор. Он выдаёт результаты для новых картинок, которых не было в базе. Их можно загрузить в систему краудсорсинга для разметки. Краудсорсеры находят ошибки, и размеченные данные доливаются в исходную базу. Процедуру можно повторять многократно, наращивая таким образом базу.

5.4.4. Человек помогает машине

В контексте взаимодействия человека и алгоритмов стоит упомянуть класс задач, в которых человек приходит на помощь алгоритму компьютерного зрения, чтобы исправить его ошибки.

Например, в задаче модерации алгоритм выдаёт в качестве результата степень уверенности. В случаях, когда алгоритм уверен, он сразу используется в системе, иначе данные подаются на ручную модерацию. В таких случаях компьютерное зрение используется для облегчения задачи модераторов, чтобы небольшое число людей могли размечать изображения.

Другой пример использования человеческого труда для улучшения качества системы — это мобильное приложение Camfind. Оно позволяет сфотографировать любой предмет и получить в ответ, что это такое.

Приложение работает в реальном времени, задержки составляют десятки секунд. Оно показывает высокое качество ответа на вопрос, что изображено на картинке. Если не знать, что приложение использует не только алгоритм, но и человеческий труд, это выглядит, как чудо. Создатели приложения долго работали над тем, чтобы люди могли быстро и качественно описывать изображение, в том числе, тщательно проработали интерфейс. С помощью приложения авторы смогли собрать большую базу картинок для своих нужд.

Подобные гибридные системы помогают улучшить качество исходного алгоритма, размечать базу, и снова улучшать качество автоматической части.

5.4.5. Синтетические изображения

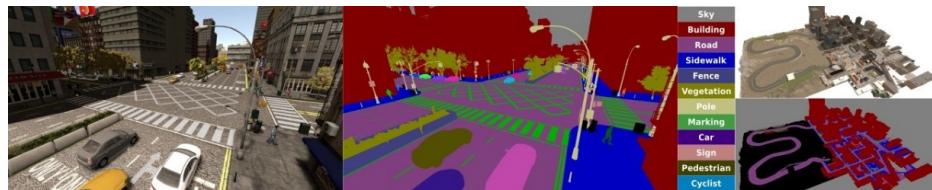


Рис. 5.21: Изображения из базы Synthia

Ещё один способ пополнения базы — это использование синтетических данных. Современная компьютерная графика позволяет генерировать реалистичные миры. Часто графические сцены выглядят реалистичнее, чем снятые в реальности. Возникает очевидна мысль использовать их для обучения классификаторов.

Примером использования такого подхода является база Synthia (рис. 5.21). Для её создания был взят искусственный город, и с помощью движка Unity были сгенерированы картинки с семантической разметкой. Эта база позволила существенно расширить существующие базы по семантической сегментации и заметно повысить качество обучения.

Использовать синтетические данные нужно аккуратно. Может сложиться так, что они отличаются от реальных, а нейронная сеть будет обращать внимание именно на эти отличия. Сгенерировать данные так, чтобы они не отличались от реальных, сложно, поэтому иногда синтетические данные никак не улучшали качество работы алгоритма. Их стоит использовать тогда, когда реальные данные действительно сложно разметить. Иначе может сложиться ситуация, когда создание системы, генерирующей синтетические данные, стоит дороже, чем ручная разметка данных.

5.4.6. Резюме

Часто наличие какой-либо базы являлось триггером для запуска исследований в той или иной области. Например, благодаря коллекции ImageNet началось современное развитие нейронных сетей. Также можно привести в качестве примера задачи распознавания лиц, детекции объектов и семантической сегментации. Во всех них начался прогресс с появления качественной коллекции.

Базы изображений могут являться конкурентным преимуществом. Корпорации делятся алгоритмами решения задач машинного обучения, выкладывают библиотеки в open source. С данными всё обстоит сложнее. Получить данные, принадлежащие какой-либо компании, практически невозможно.

Кроме того, увеличение размера и качества базы — это часто самый простой способ улучшить качество решения задачи. Вместо усложнения алгоритмов, использования ансамблей, усреднения результатов различных обучений можно просто увеличить выборку.

Урок 6

Текстовые данные и работа с ними

6.1. Работа с текстовыми данными

Эта неделя посвящена работе с текстами. Будут рассмотрены особенности текстовых данных, способы извлечения из них признаков, типичные постановки задач, а также несколько прикладных кейсов.

6.1.1. Примеры задач

Текстовые данные встречаются повсюду. В частности, можно сказать, что интернет — это огромная коллекция текстов различных форматов. С их помощью решают большое количество задач:

- предсказание по тексту записи в блоге её рейтинга. Успешность записи можно оценить ещё до публикации;
- определение эмоционального окраса комментария (положительный или отрицательный). Это может пригодиться при анализе отзывов о банке от клиентов, при обнаружении отрицательного отзыва появляется возможность быстро предпринять какие-либо действия и повысить лояльность клиента;
- определение тематики научной статьи при добавлении её в библиотеку. После этого можно либо автоматически отнести текст к выявленной категории, либо предложить автору варианты выбора;
- кластеризация новостей по сюжетам. Об одном и том же событии пишут разные новостные сайты, и имеет смысл группировать новости с одинаковым содержанием;
- поиск слов, похожих по смыслу на данное. Например, слова «стул» и «кресло» немного отличаются по смыслу, но всё-таки означают примерно одно и то же. Для компьютера оба слова — это набор символов, и с помощью простых алгоритмов нельзя определить, что это синонимы. О том, как делать такие выводы из данных, будет рассказано позднее.

Существуют и более сложные задачи на текстовых данных:

- выделить все упоминания имён в тексте. Это задача выделения новых сущностей;
- построение краткой аннотации текста;
- создание модели, отвечающей на вопросы. Она принимает на вход вопрос в произвольной форме и пытается найти на него ответ. При этом ответ строится по большому неструктурированному корпусу, например, википедии;
- генерация текста, похожего на заданный набор. Было бы интересно использовать, например, собрание сочинений Л.Н. Толстого и по нему построить модель, генерирующую тексты, похожие на рассказы автора. Уже сейчас эту задачу пытаются решать, и далее будет упомянуто, как это можно делать.

6.1.2. Текстовые признаки

Итак, задача — научиться использовать текстовые данные в алгоритмах машинного обучения: линейных моделях, решающих деревьях, нейронных сетях. Основная проблема заключается в том, что текст — это набор объектов переменной длины, состоящих из букв. Хочется преобразовать этот набор символов в фиксированное количество признаков.

Мешок слов

Существует набор данных под названием 20newsgroups. В нём требуется классифицировать тексты по 20 тематикам. Для примера можно рассмотреть следующий текст:

I am thinking about getting an Infiniti G20. In consumer reports it is ranked high in many catagories including highest in reliability index for compact cars. Mitsubishi Galant was second followed by Honda Accord). A couple of things though:

- 1) In looking around I have yet to see anyone driving this car. I see lots of Honda's and Toyota's.
- 2) There is a special deal where I can get an Infinity G20, fully loaded, at dealer cost (I have checked this out and the numbers match up). They are doing this because they are releasing and updating mid-1993 version (includes dual air-bags) and want to get rid of their old 1993's.

В нём встречаются названия марок автомобилей, что наводит на мысль о том, что тематика текста — машины. При этом порядок слов неважен, важно наличие ключевых слов.

Ещё один пример:

Announcing. . . Announcing. . . Announcing. . . Announcing. . .

CELEBRATE LIBERTY!
1993 LIBERTARIAN PARTY NATIONAL CONVENTION
AND POLITICAL EXPO
THE MARRIOTT HOTEL AND THE SALT PALACE
SALT LAKE CITY, UTAH
INCLUDES INFORMATION ON DELEGATE DEALS!
(Back by Popular Demand!)

The convention will be held at the Salt Palace Convention Center and the Marriott Hotel, Salt Lake City, Utah. The business sessions, Karl Hess Institute, and Political Expo are at t

В этом тексте присутствуют названия политических партий и собраний. Можно догадаться, что текст о политике. И снова для определения тематики важна не структура документа, а то, какие слова в него входят.

Именно на этом основан подход под названием мешок слов. В нём текст кодируется в виде количества вхождений каждого слова из словаря в документ. Этот метод хорошо себя зарекомендовал, и часто используется в задачах машинного обучения. Подробнее этот он будет обсуждаться в следующих видео.

6.2. Предобработка текста

В этой части речь пойдёт о том, какие преобразования имеет смысл совершить над текстом, прежде чем извлекать признаки и строить модели. В частности, будут рассмотрены два важных этапа предобработки: токенизация (разбиение текста на отдельные «слова») и нормализация (приведение слов к начальной форме).

6.2.1. Токенизация

Как уже было упомянуто выше, токенизация — это разбиение непрерывной строки на отдельные «слова». Для демонстрации этого процесса ниже приведён текст из википедии, объясняющий, что такое текст:

Текст (от лат. *textus* — «ткань; сплетение, связь, паутина, сочетание») — зафиксированная на каком-либо материальном носителе человеческая мысль; в общем плане связная и полная последовательность символов.

Видно, что этот отрывок состоит не только из букв, но и из символов: скобок, кавычек, тире. Кажется логичным удалить их, то есть избавиться от всего, что не является буквами или цифрами:

Текст (от лат. *textus*—«ткань; сплетение, связь, паутина, сочетание») — зафиксированная на каком-либо материальном носителе человеческая мысль; в общем плане связная и полная последовательность символов.

Если все их заменить на пробелы, то слова, разделённые пробелами, можно объявить отдельными токенами. Однако возникает много нюансов, например, слово «каком-либо». При замене дефиса на пробел оказывается, что есть два слова: «каком» и «либо». На самом деле это одно слово, которое было бы правильнее не разделять.

Итак, токенизация состоит из нескольких этапов. В первую очередь текст приводится к нижнему регистру. При этом, опять же, можно потерять часть информации. Например, «ООО» может являться сокращением (общество с ограниченной ответственностью), а «ооо» — просто выражением эмоций.

Следующий этап — это замена всех знаков препинания и прочих символов на пробелы. И снова на этом этапе возникает много нюансов. Как было упомянуто выше, при наличии сложных составных слов (например, «красно-чёрный») заменять в них дефис на пробел не очень разумно: может потеряться смысл слова. Кроме того, удаление знаков препинания приводит к удалению смайликов. Они могут играть большую роль при анализе твитов или записей в соцсетях, например, для определения эмоционального окраса текста.

После этого каждое слово, отделённое пробелом, объявляется отдельным токеном. Здесь тонкость заключается в том, что некоторые наборы слов должны рассматриваться как одно. Например, названия городов («Нижний Новгород»), или сокращения («к.т.н.», кандидат технических наук, это полезный термин при рассмотрении трёх букв вместе, но по отдельности они никакой информации не несут).

Стоит отметить, что токенизация не всегда проходит так просто. В некоторых языках не принято ставить пробелы. Например, в китайском предложение — это единый набор иероглифов. Чтобы разделить его на отдельные слова, нужно применять специальные алгоритмы сегментации текста, которые находят в нём разделители по некоторым правилам.

6.2.2. Нормализация

Следующий этап после токенизации — это нормализация слов в тексте, то есть приведение каждого слова к его начальной форме. Например, слово «машинное» необходимо привести к слову «машинный», «шёл» — «идти». Форма слова не всегда несёт в себе полезную информацию, при этом в задачах, где данных не очень много, хотелось бы использовать как можно меньшее число признаков. Их количество напрямую зависит от количества различных слов, поэтому было бы здорово сократить их число. Приведение слов к нормальной форме позволяет это сделать.

Существует два основных подхода к нормализации: стемминг и лемматизация.

Стемминг

Стемминг — это самый простой метод нормализации. Его суть состоит в том, что слова «стригутся»: по некоторым правилам от каждого слова отрезается его окончание. К сожалению, данный подход не всегда работает. Например, некоторые слова при изменении формы меняются целиком: «был», «есть», «будет». Понятно, что из этих трёх слов невозможно получить одно, отрезая буквы.

Лемматизация

По описанной выше причине чаще пользуются более сложным подходом, лемматизацией. В нём используется словарь, в котором уже записано большое количество слов и их форм. В первую очередь слово проверяется по словарю. Если оно там есть, то понятно, к какой форме его приводить. Иначе по определённому алгоритму выводится способ изменения данного слова, на основании него делаются выводы о начальной форме.

Этот подход работает лучше, и он подходит для новых неизвестных слов. Однако лемматизация сложнее, поэтому работает гораздо медленнее, чем стемминг.

6.3. Извлечение признаков из текста

Далее речь пойдёт об извлечении признаков из предобработанного текста.

6.3.1. Счётчики слов

Как уже отмечалось ранее, для текстов очень неплохо работает подход под названием мешок слов. В этом методе текст рассматривается как неупорядоченный набор слов.

Более формально описать этот метод можно следующим образом. Пусть всего в выборке N различных слов: w_1, \dots, w_N . В этом случае каждый текст кодируется с помощью N признаков, причём признак j — это доля вхождений слова w_j среди всех вхождений слов в документе.

Пусть выборка состоит из двух текстов:

“текст состоит из слов”,
“вхождения данного слова среди всех слов.”

Из этих текстов необходимо удалить слова «из» и «среди». Это так называемые стоп-слова, о которых позже будет рассказано подробнее.

Получается, что первый текст состоит из трёх слов, он кодируется вектором признаков, который находится в первой строке в таблице 6.1. Сумма значений всех признаков составляет единицу. Второй текст состоит из пяти слов, однако слово «слово» входит в него два раза, поэтому значение соответствующего признака составляет 0.4, для остальных признаков это значение равно 0.2 (строка 2 в таблице 6.1).

	текст	состоит	слово	вхождение	данный	все
текст 1	0.33	0.33	0.33	0	0	0
текст 2	0	0	0.4	0.2	0.2	0.2

Таблица 6.1: Частоты слов в текстах выборки

Используя такой подход, имеет смысл обращать внимание на два нюанса. Первый — это стоп-слова. Это популярные слова, встречающиеся в каждом тексте (например, предлоги или союзы) и не несущие в себе никакой информации, а только засоряющие признаки. Их стоит удалять ещё при предобработке, например, на этапе токенизации. Кроме того, имеет смысл удалять редкие слова. Если какое-то слово входит только в 1 или 2 текста, то, скорее всего, не получится его значимо учесть в модели и оценить, какой вклад оно вносит в целевую переменную.

6.3.2. TF-IDF

TF-IDF — это чуть более сложный подход к формированию вектора признаков. Как и в предыдущем методе, считается, что если слово часто встречается в тексте, и оно не является стоп-словом, то, скорее всего, оно важно. Но есть и вторая тонкость. Если слово встречается в других документах реже, чем в данном, то и в этом случае, скорее всего, оно важно для текста. По этому слову можно отличить этот текст от остальных. Если учесть описанные выше соображения, в результате получится подход TF-IDF.

Значение признака для слова w и текста x вычисляется по следующей формуле:

$$\text{TF-IDF}(x, w) = n_{dw} \log \frac{l}{n_w}.$$

Эта формула состоит из двух частей. n_{dw} — доля вхождений слова w в документ d . Если слово часто встречается в тексте, то оно важно для него, и значение признака будет выше. Второй множитель называется IDF (inverse document frequency, обратная документная частота), это отношение l , общего количества документов, к n_w , числу документов в выборке, в которых слово w встречается хотя бы раз. Если это отношение велико, слово редко встречается в других документах, значение признака будет увеличиваться. Если слово встречается в каждом тексте, то значение признака будет нулевым ($\log \frac{l}{l} = 0$).

6.4. Извлечение признаков из текста - 2

6.4.1. Учёт порядка токенов

Метод «мешок слов», о котором шла речь ранее, никак не учитывает порядок слов в документе, а лишь подсчитывает, сколько раз каждое слово встретилось в тексте.

Этот подход не очень хорош. Пусть в тексте есть слова «нравится» и «не нравится», у них противоположный смысл. «Мешок слов» никак не поможет уловить эти смыслы: будет только информация о том, что в тексте есть слово «нравится» и частица «не». Это не несёт много информации.

Более того, если в тексте учитывать не только слова, но и словосочетания, то признаковое пространство становится более обширным. При этом появляется возможность находить более сложные закономерности, используя простые модели (аналогично при добавлении новых признаков линейные модели могут находить более сложные разделяющие поверхности).

6.4.2. N-граммы

Использование n-грамм — это самый простой способ учитывать порядок слов. По сути n-грамма — это последовательность из n идущих подряд слов в тексте. Для предложения «Наборы подряд идущих токенов» существуют следующие n-граммы:

униграммы ($n = 1$): наборы, подряд, идущих, токенов;

биграммы ($n = 2$) : наборы подряд, подряд идущих, идущих токенов;

триграммы ($n = 3$) : наборы подряд идущих, подряд идущих токенов.

После того как в тексте найдены все требующиеся n-граммы, используются те же подходы, что и для мешка слов: подсчитываются счётчики, вычисляются TF-IDF, либо что-то ещё.

Чем больше n , до которого будут найдены n-граммы, тем больше признаков получится, признаконое пространство будет более богатым. При этом n — это гиперпараметр, и его увеличение может привести к переобучению. Например, если n — это длина текста, то у каждого документа будет уникальный признак, и алгоритм сможет переучиться под обучающую выборку.

6.4.3. Буквенные n-граммы

N-граммы можно использовать не только на словах. В качестве токенов можно рассматривать отдельные символы в предложении. После нахождения таких токенов для них можно также вычислять n-граммы (буквенные n-граммы). В качестве признаков, как и раньше, могут выступать счётчики или TF-IDF.

У этого подхода есть много преимуществ. Он позволяет учитывать смайлы или известные слова в незнакомых формах. Часто буквенные n-граммы используют вместе с n-граммами по словам.

6.4.4. Skip-граммы

Skip-граммы — это чуть более расширенный подход к использованию n-грамм. Более точно они называются k -skip- n -граммы, это наборы из n токенов, причём расстояние между соседними должно составлять не более k токенов. В качестве примера можно снова рассмотреть предложение «Наборы подряд идущих токенов». Для него:

Биграммы: наборы подряд, подряд идущих, идущих токенов;

1-skip-2-граммы: наборы подряд, подряд идущих, идущих токенов, **наборы идущих, подряд токенов**. В этом случае расстояние между токенами не должно превышать 1 слово, поэтому к предыдущему набору добавляются следующие пары слов: наборы идущих (между токенами 1 слово), подряд токенов (между токенами 1 слово);

6.4.5. Хэширование

Ещё один подход, часто использующийся для вычисления признаков на текстах, — это хэширование.

Пусть функция $h(x)$ принимает на вход и выдаёт некоторый хэш, причём возможное количество выходов функции составляет 2^n , n — некоторое небольшое число. Все слова x в тексте можно заменить на их хэши $h(x)$, а далее использовать их в качестве токенов и вычислять для них счётчики, TF-IDF или что-то другое.

Заменить слова на их хэши просто, тем не менее, у этого подхода есть свои преимущества. Он позволяет сократить количество признаков: можно не задумываться о том, какие слова объединить в один признак, а объединять слова, имеющие одинаковые хэши. Понятно, что это не очень умный подход, ведь слова можно было пытаться группировать по смыслу, но он неплохо работает и не требует никаких усилий. Но самое главное преимущество при использовании хэширования — отсутствие необходимости запоминать соответствие

между словами и номерами признаков. Не нужно помнить, например, что слово «токен» — это десятый признак в используемом признаковом пространстве, нужно лишь вычислить хэш, его значение и будет индексом признака слова. Это существенно упрощает хранение модели.

6.5. Обучение моделей на текстах

6.5.1. Подготовка выборки

Как уже было сказано ранее, прежде чем начать обучать модель, необходимо подготовить текст: сформировать выборку и вычислить признаки. Как правило, для этого сначала из текста удаляются слишком редкие или популярные слова (стоп-слова). Затем вычисляются признаки. Это могут быть n -граммы, skip-граммы, на которых вычисляются счётчики или TF-IDF.

Как правило, размерность полученного пространства признаков оказывается очень большой. При использовании униграмм количество признаков может составлять $10^3 - 10^4$, если добавить биграммы и триграммы, то при использовании большого корпуса текстов количество признаков может достигать $10^6 - 10^7$. Поэтому можно пробовать отбирать признаки, находить только те униграммы, биграммы и т.д., которые коррелируют с целевой переменной. Кроме того, можно понижать размерность. Например, использовать метод главных компонент для перехода в пространство меньшей размерности, в котором всё ещё будет сохраняться хорошее признаковое описание.

6.5.2. Обучение модели

Скорее всего, размерность признакового пространства не получится радикально уменьшить, и признаков будет очень много. Нужно понять, как в такой ситуации будут работать различные методы машинного обучения.

В случайному лесе используются деревья большой глубины, они строятся до тех пор, пока в каждом листе не окажется очень мало объектов. Если признаков очень много, то этот подход работает не очень хорошо: деревья будут очень глубокими, на их построение будет уходить слишком много времени, и можно просто не дождаться, когда лес достроится до нужного размера.

При использовании градиентного бустинга решающие деревья, наоборот, имеют очень небольшую глубину. Но это тоже проблема. Каждое дерево может учиться лишь небольшое подмножество признаков, в то время как частую ответ зависит от комбинации большого количества слов в документе. Поэтому для хорошей работы градиентного бустинга нужно будет использовать очень много деревьев, но даже в этом случае нет гарантии, что полученное качество будет приемлемым.

Байесовские методы обычно хорошо себя показывают при работе на текстах. В частности, работа наивного байесовского классификатора была продемонстрирована ранее в задаче детекции спама.

Чаще всего на текстовых данных используются линейные модели. Они хорошо масштабируются, могут работать с большим количеством признаков, на очень больших выборках.

6.5.3. Линейные модели и SGD

Для оптимизации линейных моделей может использоваться стохастический градиентный спуск. Он позволяет считывать объекты с диска по одному, и для каждого из них делать градиентный шаг. Более того, этот подход становится совсем прост в реализации, если используется хэширование:

```
пока не выполнен критерий останова:  
    t = следующий текст  
    для всех слов x в t:  
         $w_{h(x)} = w_{h(x)} - \alpha \nabla_{w_{h(x)}} Q(w)$ .
```

Урок 7

Продвинутые методы анализа текстов

7.1. word2vec

Этот урок посвящён компьютерным методам анализа текстов и начинается с описания подхода word2vec. Этот метод позволяет представить каждое слово в виде числового вектора.

7.1.1. Похожие слова

Слова «идти» и «шагать» — это синонимы, они имеют похожий смысл. Однако для компьютера это просто строки, причём не очень похожие: у них разная длина, разные буквы. Просто по этим двум строкам компьютер не может сказать, что они имеют одинаковый смысл.

При анализе текстов хочется уметь оценивать, насколько похожи различные пары слов. Это можно сделать, используя текстовые данные. Оказывается, слова со схожим смыслом часто встречаются рядом с одними и теми же словами. Другими словами, похожие слова имеют одинаковые контексты. На этом принципе основан метод word2vec.

7.1.2. Векторные представления слов

Итак, требуется описать каждое слово w с помощью вектора \vec{w} размерности d :

$$w \rightarrow \vec{w} \in \mathbb{R}^d.$$

При этом хочется иметь компактные представления слов, то есть величина d должна быть не очень большой. Также предполагается, что похожие слова должны иметь близкие векторы (например, по евклидовой или косинусной метрике). Наконец, к числовым векторам можно применять арифметические операции: складывать, вычитать, умножать на коэффициент. Хотелось бы, чтобы эти операции имели смысл. Как будет показано далее, полученные представления будут обладать всеми вышеперечисленными свойствами.

7.1.3. word2vec

При поиске вектора \vec{w} работа будет происходить с вероятностями

$$p(w_i|w_j) = \frac{\exp(\langle \vec{w}_i, \vec{w}_j \rangle)}{\sum_w \exp(\langle \vec{w}_i, \vec{w} \rangle)}.$$

Эта функция показывает, насколько вероятно встретить слово \vec{w}_i в контексте слова \vec{w}_j , то есть рядом.

Векторные представления слов будут настраиваться так, чтобы вероятности встретить слова, находящиеся в одном контексте, были высокими. В функционал для каждого слова входят вероятности встретить его вместе с k словами до и после него:

$$\sum_{i=1}^n \sum_{j=-k}^k \log p(w_{i+j}|w_i) \rightarrow \max.$$

Оптимизировать этот функционал можно с помощью стохастического градиентного спуска.

7.1.4. Свойства представлений

Если обучиться на большом корпусе текстов, то окажется, что векторы представлений имеют много интересных свойств. В частности, если слова похожи по смыслу, то соответствующие им векторы близки по косинусной метрике.

Что касается арифметических операций, то наблюдается следующее:

$$\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}},$$

$$\vec{\text{Moscow}} - \vec{\text{Russia}} + \vec{\text{England}} \approx \vec{\text{London}}.$$

Более того, оказывается, такой подход можно использовать для перевода слов. Если обучить векторы одновременно на корпусе английских и испанских текстов, то

$$\vec{\text{one}} - \vec{\text{uno}} + \vec{\text{four}} \approx \vec{\text{cuatro}}.$$

Этот метод проигрывает методам, специально разработанным для машинного перевода, тем не менее с его помощью удается получать интересные результаты.

Наконец, от представлений отдельных слов можно попытаться перейти к представлениям текстов. Оно получается усреднением векторов всех слов, входящих в документ. С полученным признаковым описанием целого текста также можно работать.

7.1.5. word2vec и обучение с учителем

Ранее уже обсуждалось, что проблема мешка слов состоит в том, что получается пространство признаков слишком большой размерности (десятки и сотни тысяч признаков). Если использовать подход word2vec, где размер вектора равен 100, то получится признаковое описание текста, размер которого также равен 100. При таком количестве признаков становится возможным использовать любые методы машинного обучения (например, случайные леса и градиентный бустинг), а не только линейные модели.

Таким образом, используя данный подход, можно значительно сжимать представления текстов, и обучать сложные методы машинного обучения.

7.2. Рекуррентные сети

Рекуррентные нейронные сети — это еще один способ работать с текстами.

7.2.1. Мешок слов

Ранее были изучены два подхода к анализу текстов. Первый — мешок слов. В нем каждое слово — это отдельный признак. Порядок слов не учитывается, предполагается, что они независимы. Считается, что текст генерируется следующим образом. Существует некоторое распределение на словах, из него случайно выбирается слово и добавляется в текст. Могут быть и более сложные порождающие процессы, но основная идея не меняется: порядок слов никак не учитывается. Можно осуществить переход к n-граммам, skip-граммам, которые учитывают связи между словами, но общий порядок слов всё равно не принимается во внимание.

7.2.2. word2vec

Второй подход — это word2vec, обучение представлений слов. В нем для получения представления слова используется его контекст, то есть слова, рядом с которыми оно часто встречается. В этом случае появляется некоторый порядок. Однако признаковое описание текста получается усреднением признаковых описаний входящих в него слов, и порядок слов снова никак не учитывается.

7.2.3. Рекуррентная сеть

Возникает вопрос, существует ли подход, который использует текст именно в исходном виде, то есть учитывает порядок слов. Такие методы существуют, и один из них — это рекуррентные нейронные сети (рис. 7.1). Такая сеть устроена немного необычным образом. Она последовательно получается на вход слова. При этом в ней есть скрытый слой h , который обновляется после появления каждого нового слова или токена

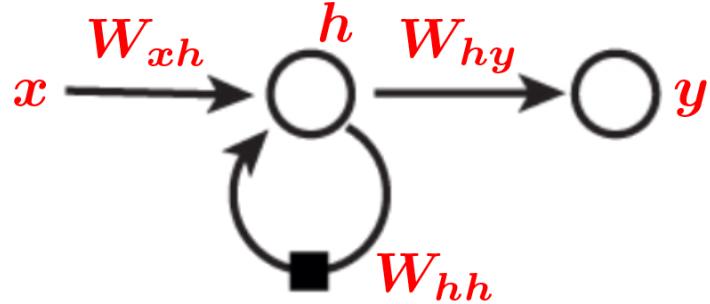


Рис. 7.1: Структура рекуррентной нейронной сети

(например, буквы). Слой обновляется, учитывая и своё предыдущее состояние, и полученное слово. После того как скрытое состояние обновлено, на его основе генерируется выход y для данного токена. Процесс повторяется до тех пор, пока через сеть не пройдут все слова в тексте.

Если говорить более строго, то данная сеть описывается двумя формулами. Первая — это обновление скрытого состояния. Оно обновляется путём некого усреднения предыдущего значения и нового входа:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1}),$$

где h_{t-1} — состояние слоя в предыдущий момент времени, x_t — слово, которое было подано на вход, f — некая нелинейная функция активации.

Вторая формула, которой описывается сеть, — это значение на выходе:

$$y_t = g(W_{hy}h_t),$$

где h_t — обновлённое состояние скрытого слоя, g — функция активации.

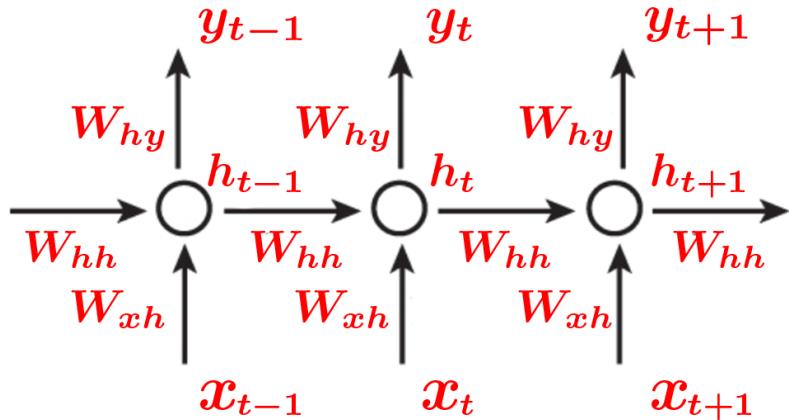


Рис. 7.2: Структура рекуррентной нейронной сети, развёрнутой во времени

Описанное выше представление является наглядным, но не очень удобным. Нейронную сеть можно изобразить в более привычном виде, если развернуть её во времени (рис. 7.2). Это обычная нейронная сеть прямого распространения, длина которой зависит от числа токенов, которые подаются на вход. При этом параметры, матрицы W_{hh} , W_{xh} , W_{hy} , являются общими и не меняются от шага к шагу. Изменяются только скрытые состояния h_t , входы x_t , выходы y_t . Этот вид сети нужен для того, чтобы её обучать.

7.2.4. Обучение

Обучать описанную выше сеть можно с помощью обратного распространения ошибки. Этот метод немного усложняется, потому что в сети между разными итерациями присутствуют общие параметры, и это нужно учитывать при дифференцировании. Также в обучении возникают особенности, связанные с тем, что градиенты распространяются очень далеко. Если нейронная сеть глубокая, в тексте много токенов, может возникнуть

затухание или, наоборот, взрыв градиентов. Эти проблемы здесь обсуждаться не будут, предполагается, что они решены в готовых пакетах для обучения нейронных сетей.

Существует много примеров использования описанной нейронной сети, далее будут рассмотрены два из них.

7.2.5. Пример: генерация текста

Порождение текстов — это, наверное, самый популярный способ использования рекуррентных нейронных сетей. На вход сети подаётся слово, а выход — это вектор вероятностей, размер которого равен размеру используемого словаря (количество слов или символов), то есть распределение на словах (или токенах). Если выбрать токен с наибольшей вероятностью, то таким образом сгенерируется новый токен. После этого можно его подать на вход, сгенерировать ещё один и т. д. Таким образом получается генерация текста.

Если обучить такую нейросеть на некотором корпусе, то можно будет генерировать похожие тексты.

Простой пример. Пусть нейронная сеть обучена на всех текстах из Википедии, причём обучение происходило на разметке этих текстов. Если сгенерировать новый текст для Википедии, то получится что-то похожее на текст, показанный на рисунке 7.3.

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servacious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS) (<http://www.humah.yahoo.com/guardian.cfm/7754800786d17551963ss89.htm>) Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile..]

Рис. 7.3: Текст, сгенерированный нейронной сетью, обученной на Википедии

Следует обратить внимание, что в этом тексте правильно сгенерирована разметка (ссылки с квадратными скобками, ссылки на веб-страницы и т.д.), то есть нейросеть хорошо улавливает структуру текста и элементы, которые в нём встречаются.

7.2.6. Пример: обучение с учителем

Ещё один пример использования рекуррентных нейронных сетей — это генерация признаков для обучения с учителем. Существует несколько подходов к тому, как извлекать из таких сетей признаки.

Первый подход заключается в том, чтобы пропустить весь текст через нейронную сеть, и в качестве признаков использовать получившийся вектор скрытого слоя. Поскольку скрытое состояние хранит в себе информацию обо всех прошедших через него словах в прошлом (если матрица переходов грамотно обучена), то полученное представление также будет содержать информацию обо всём тексте.

Можно поступить иначе: извлекать скрытый вектор после каждой итерации (после подачи каждого токена на вход сети), а после — каким-то образом их все агрегировать (например, усреднить).

Урок 9

Ранжирование

9.1. Задача ранжирования

Этот урок посвящён задаче ранжирования. Она имеет много общего с уже рассмотренными ранее задачами классификации и регрессии, но есть и ряд особенностей, которые и будут рассмотрены в данном уроке.

9.1.1. Ранжирование

Формальная постановка задачи ранжирования выглядит следующим образом. Имеется выборка, состоящая из l элементов:

$$x_1, \dots, x_l.$$

Она состоит из объектов, имеющих признаковое описание, как и в задачах регрессии и классификации.

Ключевое отличие ранжирования заключается в целевой переменной. Если ранее в задачах обучения с учителем каждому объекту соответствовал свой ответ, то теперь ответы — это пары вида:

$$\{(i, j) : x_i < x_j\}.$$

Набор таких пар и является целевой переменной.

В задаче требуется построить ранжирующую модель $a(x)$, такую, что:

$$x_i < x_j \Rightarrow a(x_i) < a(x_j).$$

Таким образом, по выходам необходимо восстановить порядок, а величина ответов не имеет значения. В этом заключается главное отличие ранжирования от остальных задач машинного обучения.

9.1.2. Ранжирование поисковой выдачи

Основное применение ранжирования — это ранжирование поисковой выдачи. Например, в Яндексе в качестве объектов выступают пары запрос, документ. Запрос — это ключевые слова, введённые пользователем, документ — один из всех документов, имеющихся в поисковом индексе:

$$(\text{запрос}, \text{документ}_1) < (\text{запрос}, \text{документ}_2).$$

Задача — научиться восстанавливать порядок на таких парах, причём запрос во всех парах один и тот же. Таким образом, требуется ранжировать документы для одного запроса.

Для обучения такой порядок задаётся ассессорами. Это специальные люди, которые получают пары запрос, документ и оценивают релевантность документа, то, насколько хорошо он отвечает данному запросу. Оценки могут быть как числовыми, так и попарными (ассесор в паре документов выбирает тот, который более релевантен запросу).

9.1.3. Рекомендации

Другой пример применения ранжирования — это рекомендации. В этой задаче по парам пользователь, товар нужно ранжировать товары для данного пользователя, максимизируя при этом некоторую метрику. Порядок задаётся на основании предпочтений пользователя, его предыдущих покупок и оценок.

9.1.4. Особенности задачи

У задачи ранжирования есть свои особенности. Во-первых, объекты не являются независимыми: целевая переменная зависит от пар объектов. Это необходимо учитывать при решении. Во-вторых, используются более сложные метрики, чем для регрессии или классификации. Обычно они дискретные, потому что важны не получаемые значения, а порядок, который задаёт модель. Ещё одна особенность — для этой задачи очень важно правильно сформировать выборку, потому что это можно сделать множеством способов. Непонятно, например, какие именно пары отдавать на разметку асессорам в задаче ранжирования поисковой выдачи, или как именно получать предпочтения пользователей из данных для задачи построения рекомендаций.

9.2. Метрики качества ранжирования

9.2.1. Точность ранжирования

Для простоты всюду далее в качестве примера будет использоваться задача ранжирования поисковой выдачи, но все рассуждения хорошо обобщаются и для других применений.

В самой простой постановке задачи ранжирования целевая переменная принимает два значения, документ либо релевантен запросу, либо нет:

$$y(q, d) \in \{0, 1\},$$

где y — целевая переменная, q — запрос, d — документ.

Пусть также есть некоторая модель $a(q, d)$, оценивающая релевантность документа запросу. По значениям, полученным с помощью этой модели, можно отранжировать документы. $d_q^{(i)}$ будет обозначать i -й по релевантности документ для запроса q .

После того как введены обозначения, можно задать простейшую метрику ранжирования. Это Precision@ k , точность среди первых k документов (k — параметр метрики). Если ранжируется поисковая выдача, и на первой странице показываются 10 документов, то разумно выбирать $k = 10$. Данная метрика определяется как доля релевантных документов среди первых k , полученных с помощью модели:

$$\text{Precision}@k(q) = \frac{1}{k} \sum_{i=1}^k y\left(q, d_q^{(i)}\right),$$

Это полезная метрика, потому что обычно важно иметь релевантные документы среди первых k . Однако у неё есть серьёзный недостаток: позиции релевантных документов никак не учитываются. Например, если при $k = 10$ среди первых k документов есть 5 релевантных, то неважно, где они находятся: среди первых или последних 5 документов. Обычно же хочется, чтобы релевантные документы располагались как можно выше.

Описанную проблему можно решить, модифицировав метрику, и определить среднюю точность (average precision, AP). Данная метрика тоже измеряется на уровне k и вычисляется следующим образом:

$$\text{AP}@k(q) = \frac{\sum_{i=1}^k y\left(q, d_q^{(i)}\right) \text{Precision}@i(q)}{\sum_{i=1}^k y\left(q, d_q^{(i)}\right)}.$$

Данная величина уже зависит от порядка. Она достигает максимума, если все релевантные документы находятся вверху ранжированного списка. Если они смещаются ниже, значение метрики уменьшается.

И точность, и средняя точность вычисляются для конкретного запроса q . Если выборка большая и размечена для многих запросов, то значения метрик усредняются по всем запросам:

$$\text{MAP}@k = \frac{1}{|Q|} \sum_{q \in Q} \text{AP}@k(q).$$

9.2.2. DCG

Второй подход к измерению качества ранжирования — это метрика DCG (discounted cumulative gain). Она используется в более сложной ситуации, когда оценки релевантности y могут быть вещественными:

$$y(q, d) \in \mathbb{R}.$$

То есть для каждого документа теперь существует градация между релевантностью и нерелевантностью. Остальные обозначения остаются теми же, что и для предыдущей метрики.

Формула для вычисления DCG:

$$\text{DCG}@k(q) = \sum_{i=1}^k \frac{2^{y(q, d_q^{(i)})} - 1}{\log(i+1)}.$$

Метрика — это сумма дробей. Чем более релевантен документ, тем больше числитель в дроби. Знаменатель зависит от позиции документа, он штрафует за то, где находится документ. Если документ очень релевантен, но занимает низкую позицию, то штраф будет большим, если документ релевантен и находится вверху списка, штраф будет маленьким. Таким образом, метрика DCG учитывает и релевантность, и позицию документа. Она достигает максимума, если все релевантные документы находятся в топе списка, причём отсортированные по значению y .

Данную метрику принято нормировать:

$$n\text{DCG}@k(q) = \frac{\text{DCG}@k(q)}{\max \text{DCG}@k(q)},$$

где $\max \text{DCG}@k(q)$ — значение DCG при идеальном ранжировании. После нормировки метрика принимает значения от 0 до 1.

9.3. Методы ранжирования

Всего выделяют три подхода к решению задачи ранжирования: pointwise (поточечный), pairwise (попарный), listwise (списочный). Далее будут приведены по одному методу из каждого подхода, чтобы можно было составить представления об их различиях и особенностях.

9.3.1. Поточечный подход

Самый простой подход — это поточечный. В нём игнорируется тот факт, что целевая переменная $y(q, d) \in \mathbb{R}$ задаётся на парах объектов, и оценка релевантности $a(d, q)$ оценивается непосредственно для каждого объекта.

Если речь идёт о задаче ранжирования, то пусть ассессор поставил какую-то оценку y каждой паре запрос, документ. Эта оценка и будет предсказываться. При этом никак не учитывается, что на самом деле нужно предсказать порядок объектов, а не оценки. Этот подход является простым в том смысле, что в нём используются уже известные методы. Например, можно предсказывать оценки с использованием линейной регрессии и квадратичной ошибки:

$$\sum_{i=1}^{\ell} (a(q_i, d_i) - y(q_i, d_i))^2 \rightarrow \min.$$

Известно, как решать такую задачу, и таким образом будет получена релевантность. Далее по выходам модели можно ранжировать объекты.

9.3.2. Попарный подход

В попарном подходе используются знания об устройстве целевой переменной. Модель строится минимизацией количества дефектных пар, то есть таких, в которых моделью был предсказан неправильный порядок:

$$\sum_{x_i < x_j} [a(x_j) - a(x_i) < 0] \rightarrow \min.$$

К сожалению, этот функционал дискретный (в него входят индикаторы), поэтому не получится минимизировать непосредственно его. Однако можно действовать так же, как и с классификаторами: оценить функционал сверху.

Можно считать, что разница между объектами $a(x_j) - a(x_i)$ — это отступ M , и задать некоторую гладкую функцию $L(M)$:

$$\sum_{x_i < x_j} L(a(x_j) - a(x_i)) \rightarrow \min.$$

Если использовать функцию как в логистической регрессии $L(M) = \log(1 + e^{-M})$, то полученный метод называется RankNet. Затем можно решать задачу, например, с помощью стохастического градиентного спуска.

9.3.3. Listwise-подход

В методе RankNet шаг стохастического градиентного спуска для линейной модели выглядит следующим образом:

$$w := w + \eta \frac{1}{1 + \exp(\langle w, x_j - x_i \rangle)} (x_j - x_i).$$

Это не очень сложная формула, она зависит от одной пары объектов. Возникает вопрос, можно ли модифицировать данный метод (а именно формулу шага) так, чтобы минимизировался не исходный функционал, оценивающий долю дефектных пар, а DCG.

Ответ на этот вопрос положительный. Можно домножить градиент исходного функционала на то, насколько изменится NDCG, если поменять местами x_i и x_j :

$$w := w + \eta \frac{1}{1 + \exp(\langle w, x_j - x_i \rangle)} (x_j - x_i) \cdot |\Delta \text{NDCG}_{ij}| (x_j - x_i).$$

Оказывается, что при выполнении градиентного спуска с помощью данных шагов оптимизируется NDCG. Это эмпирический факт, и он не доказан. Но на практике NDCG действительно улучшается при решении задачи данным методом.

Существуют и другие подходы к оптимизации NDCG, однако в них предпринимается попытка работы с функционалом, что гораздо сложнее. Выше описан самый простой подход, он называется LambdaRank.