**8. МАШИННОЕ ОБУЧЕНИЕ В АНАЛИЗЕ ВРЕМЕННЫХ РЯДОВ**

Рассмотрим несколько примеров применения методов машинного обучения к анализу временных рядов. Это относительно новая дисциплина науки о данных, но она демонстрирует многообещающие результаты. В отличие от статистических моделей, которые мы изучали ранее, методы машинного обучения, с которыми нам предстоит познакомиться, изначально не разрабатывались для анализа временных рядов, но оказались очень полезными для работы с ними.

Переход к технологиям машинного обучения задуман как продолжение изысканий по составлению прогнозов, проводимых ранее. До настоящего момента наше внимание было сконцентрировано на статистических моделях прогнозирования временных рядов. При разработке таких моделей мы сформулировали основную теорию описания динамики временного ряда и статистических способов представления шума и неопределенности в его поведении. Впоследствии на основе рассчитанной гипотетической динамики процесса были составлены прогнозы и оценена степень их неопределенности. При таком подходе идентификация и оценка параметров модели требовали предельно тщательной проработки — они должны наилучшим образом описывать динамику изменения данных.

Здесь рассматриваются вычислительные методики, в которых не учитываются характеристики базового процесса и правила, описывающие его поведение. Вместо этого в них поведение системы описывается согласно шаблонам, которые распознаются с помощью методов, сходных с применяемыми для предсказания целевой величины, например, классификационных подписей в данных временного ряда. В ней мы также рассмотрим методы обучения без учителя на примере кластеризованных данных временных рядов.

Здесь для описания принципов прогнозирования и классификации применяются древовидные модели, а кластеризация представлена как специальная форма классификации. В методах классификации деревьев крайне важно определиться с признаками временного ряда, поскольку в них, в отличие, например, от ARIMA, не учитываются временные изменения в поведении данных.

В случае кластеризации и метрической классификации будет показано, что в качестве входных данных можно использовать как признаки, так и исходные временные ряды. Чтобы подать на вход сам временной ряд, нужно рассчитать метрику расстояния, известную как динамическая трансформация времени, которая в случае применения к временным рядам позволяет сохранить в данных полный хронологический набор информации, не прибегая к ее представлению ограниченным набором признаков.

***Классификация временных рядов***

Рассматривается пример представления признаками необработанных временных рядов данных электроэнцефалограммы (ЭЭГ), которые в дальнейшем подвергаются обработке с помощью алгоритмов машинного обучения.

После извлечения признаков из временного ряда ЭЭГ для классификации данных обратимся к деревьям решений.

***Выбор и генерация признаков***

Ранее мы посвятили много времени, описывая цели, преследуемые при генерации признаков временных рядов. Кроме того, рассматривался небольшой пример получения признаков для данных временных рядов с помощью пакета tsfresh. В этой части признаки для временных рядов будут генерироваться с помощью другого, также обсуждаемого ранее, программного пакета cesium.

Одно из главных преимуществ пакета cesium заключается в его снабжении множеством полезных наборов временных рядов, включая данные ЭЭГ, первоначально полученных в исследовательской работе 2001 года. В ней детально описаны все операции по подготовке данных. Нам же достаточно знать, что все пять категорий временных рядов ЭЭГ из встроенного набора данных представляют собой фрагменты равной длины, вырезанные из непрерывных временных отсчетов, избранных ЭЭГ. В частности, среди них вы найдете следующие временные ряды.

• ЭЭГ-записи здоровых людей с открытыми и закрытыми глазами (две отдельные категории)

• ЭЭГ-записи пациентов с эпилепсией в периоды без судорог для двух областей мозга, не связанных с проявлением судорог (две отдельные категории)

• Внутричерепная запись ЭЭГ во время судорог (одна категория) Набор данных загружается с помощью специальной функции, также включенной в пакет cesium.

## Python

from cesium import datasets

eeg = datasets.fetch\_andrzejak()

Всегда полезно предварительно просмотреть несколько примеров анализируемых данных, чтобы получить представление о том, как такие временные ряды лучше всего классифицировать.

#Python

pit.subplot(3, 1, 1)

pit.plot (eeg[ "measurements"] [0])

pit.legend(eeg['classes' ] [0])

##Z

pit.subplot(3, 1, 2)

pit.plot (eeg["measurements”] [300])

pit.legend(eeg['classes' ] [300] )

##F

pit.subplot (3, 1, 3)

pit.plot(eeg["measurements"] [450])

pit.legend(eeg['classes' ] [450])

##S

На графиках заметны отчетливые различия между классами ЭЭГ (рис. 8.1). И это не удивительно: графики ЭЭГ указывают на активность разных частей мозга при выполнении разных действий — как у здоровых субъектов, так и у пациентов с эпилепсией. Результат визуализации может послужить руководством к генерации признаков.

Рис. 8.1. Графики трех случайных выборок из набора данных ЭЭГ. Это выборки независимых измерений, а не результаты исследований, проводимых одновременно в разных частях одного и того же мозга. Представленные данные получены для разных пациентов в разное время

В частности, классы Z и F, скорее всего, содержат меньше искаженных данных, чем класс S. Кроме того, как видно по оси у, каждый класс характеризуется своим, отличным от остальных, диапазоном значений. Это указывает на то, что в качестве признака может использоваться амплитуда. Помимо того, графики различаются не только амплитудой значений, но и общим распределением точек, которые различаются у всех трех классов. Таким образом, в дальнейшем анализе мы будем опираться на эти и некоторые другие признаки, после чего напишем код их генерации.

***Визуализация и машинное обучение***

Данные ЭЭГ отлично визуализируются на двумерной гистограмме. В частности, такая гистограмма предельно точно представляет данные каждого типа временных рядов и подтверждает (или опровергает) репрезентативность исследуемых признаков. В нашем случае она позволяет установить, будет ли амплитуда или наклон прямой устойчиво различаться для разных классов.

Кроме того, нелишне построить одномерные гистограммы — по одной на класс. Также рассмотрите возможность проведения дополнительных исследований, заключающихся в ядерной оценке плотности ([ru.wikipedia.org/wiki/Ядерная\_оценка\_плотности](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BE%D1%86%D0%B5%D0%BD%D0%BA%D0%B0_%D0%BF%D0%BB%D0%BE%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B8)) для каждого класса.

Визуально ядерная оценка плотности схожа с гистограммой, но строится так, чтобы обладать специальными свойствами, такими как непрерывность и гладкость, эффективно приводя необработанные данные гистограммы к плавному, непараметрическому оценочному распределению для базовой переменной. Википедия предлагает хорошее описание ядерной оценки плотности, которая может быть выполнена с помощью таких программных инструментов, как пакет scikit-learn в языке Python или функция density() пакета stats в языке R.

Сгенерируем признаки с помощью пакета cesium.

## Python

from cesium import featurize. featurize\_time\_series as ft

eatures\_to\_use = ["amplitude",

"percen t\_beyond\_l\_s td ",

"percent\_close\_to\_median",

"skew",

"max\_slope"]

fset\_cesium = ft (times = eeg ["times"],

values = eeg["measurements"],

errors = None,

features\_to\_use = features\_to\_use,

scheduler = None)

Этот код создает признаки, представленные в таблице рис. 8.1 (снимок экрана виртуального ноутбука Jupyter).

Рис. 8.2. Числовые значения признаков, сгенерированных для первых нескольких выборок анализируемого набора данных

Заметьте, что многие значения не нормализованы. Это следует учитывать при использовании методов, которые требуют передачи исключительно нормализованных входных данных.

Нужно четко понимать, на что именно указывают признаки, и уметь сопоставлять ожидания с результатами, полученными с помощью пакета cesium.

Чтобы проиллюстрировать важность проверки полученных результатов и отсутствия противоречий со здравым смыслом, давайте проанализируем параметр percent\_beyond\_1\_std для единственной выборки временного ряда.

## Python

>>>np. std (eeg\_small ["measurements"] [0])

40.411

>>>np.mean (eeg\_small ["measurements"] [0])

-4.132

>>>sample\_ts = eeg\_small ["measurements"] [0]

>>>sz = len(sample\_ts)

>>>ll = -4.13 - 40.4

>>>ul = -4.13 + 40.4

>>>quals = [i for i in range(sz) if sample\_ts[i] < ll or

sample\_ts[i] > ul]

>>>len (quals)/len (ser)

0.327 ## Проверка сгенерированного признака (см, рисунок)

***Эргодические признаки***

При выборе признаков временного ряда из автоматически сгенерированных вариантов обращайте внимание только на эргодические признаки. Они предполагают, что каждое из измерений будет сходиться к устойчивому значению по мере сбора большего на графике, различные подвыборки из любого заданного временного ряда явно сопоставимы, а сам ряд слабо стационарен, поэтому сгенерированные признаки имеют смысл.

Необходимо иметь возможность проверки используемых признаков. Это вопрос ответственного анализа. Не стоит предоставлять алгоритму информацию, которую нельзя понять, объяснить и проверить.

Не злоупотребляйте библиотеками, предназначенными для генерации признаков. Написать собственный код генерации признаков совсем не трудно. Если вы работаете с данными предметной области, в которой определенный набор признаков используется неоднократно и в одних и тех же комбинациях, то постарайтесь написать собственный код реализации алгоритма (даже если исходно использовали инструменты программных библиотек).

Вы сможете оптимизировать собственное решение намного лучше, чем разработчики равнозначного универсального пакета. Например, располагая несколькими признаками, основанными на вычислении среднего значения временного ряда, можно создать код однократного вычисления такого значения и не дублировать его для каждого отдельно рассчитываемого признака.

***Деревья принятия решений***

Деревья принятия решений имитируют принятые у людей способы принятия решений — пошагово и в высшей степени нелинейно. Они отражают принципы принятия нами сложных решений: одно за раз, изучая то, как текущая переменная влияет на полученное решение, затем учитывается следующий вариант — очень похоже на блок-схему.

С большой вероятностью вам уже доводилось работать с деревьями принятия решений или вы знакомы с их общей концепцией. Если это не так, то поставьте изучение материала на паузу, сделайте перерыв и ознакомьтесь с основными справочными сведениями о деревьях принятия решений (<https://perma.сс/G9AA-ANEN>).

На рисунке 8.3 показан простой пример дерева принятия решений, которое можно использовать для оценки веса тела любого человека.

Городской или сельский?

Мужчина или женщина?

Ребенок или взрослый?

18 кг

76 кг

55 кг

63 кг

Рис. 8.3. В этом простом дереве ряд логических ветвей используется для предсказания веса человека. Это грубая модель, но она иллюстрирует нелинейный и изменчивый подход, который можно реализовать в регрессионной задаче с помощью простой модели дерева принятия решений

Существует множество примеров человеческого поведения, следующего принципам дерева принятия решений при анализе данных временных рядов. Например, независимые трейдеры на фондовом рынке также получают доступ к техническим индикаторам, но, скорее всего, изучают их пошагово в некой иерархически выверенной манере, подобно тому, как это принято в дереве нелинейных решений — сначала определяя направление движения рынка по одному техническому индикатору, а затем, получив один из вариантов ответа, исследуя для него временные изменения волатильности. По всей видимости, они удерживают в памяти некое подобие древовидной структуры принятия решений, которую они используют, чтобы составлять достаточно точные прогнозы о направлении движения рынка.

Подобным образом медицинские сотрудники читают ЭЭГ и ЭКГ. Сначала они ищут хотя бы один из известных признаков, и только затем переходят к поиску следующего, последовательно учитывая все возможные факторы. При наличии только одного признака и отсутствии другого диагноз будет отличаться от утверждаемого в противном случае и, следовательно, другого прогноза относительно состояния пациента.

Далее рассмотрим признаки, сгенерированные для данных ЭЭГ, применяемых в качестве входных данных в двух разных методах дерева принятия решений — случайного леса и градиентного бустинга деревьев, — каждый из которых используется в задачах классификации. Наша задача будет заключаться в классификации исследуемых данных ЭЭГ исключительно на основе признаков, сгенерированных из необработанных данных.

***Случайный лес***

Случайный лес — это модель, в которой используется не одно, а множество деревьев принятия решений. Результатом усреднения выходных данных таких деревьев будет классификация или регрессия данных. Случайные леса основаны на понятии “мудрости толпы”, где “толпа” представлена большим количеством простых моделей, ни одна из которых сама по себе не может быть достаточно хороша, но все вместе они часто превосходят даже самое точное, но единственное дерево решений.

Идея собрать коллекцию моделей для составления точного прогноза, а не просто найти единственную “лучшую” модель, была высказана еще в 1969 году в статье Combination of Forecasts (Комбинации прогнозов) — исследовательской работе двух именитых статистов, Дж. М. Бейтса и С. В. Дж. Грейнджер. В этом документе показано, что объединение двух отдельных прогнозов данных о пассажирах авиакомпании может привести к получению модели с более низкой среднеквадратичной ошибкой, что оказывается совершенно неожиданным и в то же время неубедительным результатом. Тем не менее молодое поколение аналитиков, часто прибегающих при исследовании данных к услугам алгоритмов машинного обучения (в противоположность статистическим методам), наоборот, считают такой подход интуитивно понятным, а не обескураживающим. Очень быстро случайный лес деревьев стал наиболее частым решением, применяемым к большинству задач прогнозирования.

Случайный лес строится в соответствии с параметрами, определяющими количество деревьев для обучения, а также максимально допустимую глубину этих деревьев. Впоследствии обучение каждого отдельного дерева выполняется на случайной выборке данных и их признаков. Деревья, как правило, параметризуются так, чтобы получить максимально простую структуру и избежать эффекта переобучения, а модель должна уметь усреднять многие другие модели, ни одна из которых не является особенно хорошей, но все они достаточно общие, чтобы избежать “ловушек”, таящихся в данных. Как упоминалось ранее, признаки, рассчитанные для каждой выборки временного ряда, вводятся в модель в качестве результатов обучения. Теоретически можно придумать способы ввода в модель не вычисленных признаков, а необработанных временных рядов, но их разработка связана с рядом трудностей.

• Временные ряды неодинаковой длины обрабатывать очень сложно.

• Использование большого количества входных данных (равного или близкого к числу временных шагов) приведет к созданию невероятно сложных и дорогостоящих для вычисления, а также обучения моделей.

• Если предположить, что ни один временной шаг не является важным (поскольку каждый временной шаг соответствует хотя бы одному признаку), то модель будет характеризоваться высоким уровнем шума и низким уровнем полезного сигнала для обучения, что будет проявляться на каждом шаге ввода данных.

Таким образом, случайные леса не являются хорошим инструментом для исследования данных временных рядов в необработанном виде, но становятся полезными после сжатия данных в сводные признаки. Перечислим несколько подтверждающих факторов.

• С точки зрения эффективности/вычислительных затрат принято считать, что очень длинные временные ряды всегда можно представить набором признаков и подобрать для них модель с разумной точностью.

• Очень важно добиться того, чтобы случайный лес понижал риск переобучения.

Как обсуждалось ранее, переобучение является особенно частой проблемой анализа временных рядов из-за наличия выраженной взаимосвязи между методами обучения и прогнозирования. Преднамеренно упрощенная методология позволяет устранить некоторые из потенциальных трудностей.

• Случайные леса могут быть особенно подходящими для данных временных рядов, для которых не предусмотрена рабочая модель или гипотеза о механике базового процесса.

Специалистам по анализу удалось добиться заметных успехов только в развертывании случайных лесов в задачах классификации временных рядов, но не в прогнозирования данных. В противоположность им рассматриваемый в следующем разделе метод, градиентный бустинг деревьев, получил широкое распространение при решении задач обоих типов.

***Градиентный бустинг деревьев***

Бустинг является еще одним способом построения ансамбля предикторов. Бустинг, предполагающий последовательный способ построения модели, основан на предположении, что более поздние модели исправляют ошибки более ранних моделей, а несоответствие данных в более ранних моделях должно в большей степени учитываться и устраняться более поздними моделями.

Градиентный бустинг деревьев оказался особенно успешным методом исследования временных рядов, о чем свидетельствует его популярность среди участников соревнований по анализу данных, проводимых в течение нескольких последних лет.

Библиотека XGBoost строит деревья принятия решений последовательным образом так, что каждое следующее дерево стремится предсказать остатки комбинации предыдущих деревьев. Так, например, первое дерево, построенное библиотекой XGBoost, будет пытаться сопоставить данные напрямую (категория или числовое значение). Второе дерево будет предсказывать истинное значение за вычетом прогнозируемого значения. Третье дерево будет предсказывать истинное значение минус предсказанное значение первого дерева минус предсказание второго дерева об остатках первого дерева.

Однако библиотека XGBoost, пытаясь минимизировать остатки прогнозируемых остатков прогнозируемых остатков, и так до бесконечности, не стремится создать неисчислимое количество моделей. Алгоритм XGBoost минимизирует функцию потерь, которая также включает штрафной член для сложности модели, и именно этот штрафной член ограничивает количество получаемых деревьев.

Кроме того, количество производимых деревьев можно указать напрямую. За последние несколько лет появилось много извещений о больших успехах использования пакета XGBoost не только в методах машинного обучения, но и методах анализа временных рядов. В частности, они поступали с соревнований Kaggle и отраслевых конференций по машинному обучению.

*Баггинг* (или, более формально, бустстрэп-агрегирование) — это метод обучения моделей, при котором для каждой отдельной модели в ансамбле создаются случайно сгенерированные обучающие наборы. Случайные леса, как правило, используют методологию баггинга при обучении моделей.

*Бустинг*, как отмечалось выше, относится к методу обучения моделей, в которых ансамбль состоит из последовательно обучаемых моделей, каждая из которых направлена на исправление ошибки, допущенной ее предшественницей. Градиентный бустинг лежит в основе обучения деревьев.

***Пример кода***

В случае применения случайных лесов модель машинного обучения проще кодировать с помощью инструментов библиотеки XGBoost, чем разбираться в том, как она работает. В следующем примере мы будем обучать модель случайного леса и использовать градиентный бустинг для классификации исходных данных ЭЭГ на основе предварительно сгенерированных признаков. Для разделения данных на обучающий набор и набор тестирования воспользуемся пакетом sklearn.

## Python

from sklearn,model\_selection import train test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

fset\_cesium.values, eeg["classes"], random\_state=21)

Начнем с создания классификатора, основанного на случайном лесе. Как видите, модель классификации данных ЭЭГ реализуется невероятно просто.

## Python

from sklearn.ensemble import RandomForestClassifier

rf\_clf = RandomForestClassifier(n\_estimators = 10,

max\_depth = 3,

random\_state = 21)

rf\_clf.fit(X\_train, y\_train)

Теперь нужно определить точность данных за пределами выборки, вызвав метод объекта Classifier.

## Python

>>>rf\_clf. score (X\_test, y\_test)

0.616

Всего несколько строк кода реализуют модель намного лучше и быстрее, чем это может сделать человеческий мозг (лишенный медицинского образования). Помните, что благодаря правильному выбору признаков модель работает со сводной статистикой, а не с полными данными ЭЭГ. Код классификатора XGBoost также предельно прост и выразителен.

## Python

>>>import xgboost as xgb

>>>xgb\_clf = xgb.XGBClassifier(n\_estimators = 10,

max\_depth = 3,

random\_state = 21)

>>>xgb\_clf. fit (X\_train, y\_train)

>>>xgb\_clf. score (X\_test, y\_test)

0.648

Легко заметить, что модель классификатора XGBoost работает немного лучше, чем модель случайного леса. Она также обучается немного быстрее, что показывает следующий эксперимент по расчету времени обучения каждой модели.

## Python

>>>start = time, time ()

>>>xgb\_clf. fit (X\_train, y\_train)

>>>end = time, time ()

>>>end - start

0.0189

## Случайный лес

>>>start = time.time ()

>>>rf\_clf. fit (X\_train, y\_train)

>>>end = time, time ()

>>>end – start

0.027

Скорость проведения вычислений существенно повысилась: модель случайного леса рассчитывается на 50% дольше, чем XGBoost. Хотя это не окончательный тест, он все же указывает на преимущество XGBoost, особенно при работе с большими наборами данных. Убедитесь в том, что это преимущество увеличится при использовании больших наборов данных с большим количеством выборок и признаков.

***Это не тестирование производительности!***

Можно высказать много критических замечаний по поводу предыдущего кода. Для начала заметим, что существуют более простые варианты реализации методов тестирования производительности. Кроме того, время обучения необходимо усреднять, а код запускать на реальных данных в не менее реальных условиях обучения, а не на примитивном примере, существующем исключительно в виртуальном ноутбуке Jupyter.

Помимо того, низкая скорость работы алгоритма общего назначения на неких данных совсем не означает, что алгоритм имеет низкую производительность. Зачастую общедоступные программные модули исходно пишутся так, чтобы, в первую очередь, обеспечить прозрачность кода и только затем — достойную производительность.

В таких случаях алгоритм действительно работает медленнее, чем мог бы. В остальных случаях алгоритмы отлаживаются под как можно более широкий круг задач, тогда как вам требуется более узкоспециализированная реализация, т.е. стандартные реализации алгоритмов нужно изменять в соответствии с поставленными целями.

Вопросы производительности, характерные для временных рядов, рассматривать необходимо (но не в этом курсе).

Возникает справедливый вопрос о том, существует ли в рассматриваемом наборе гиперпараметров что-то, что предоставляет XGBoost преимущество над случайным лесом. Например, что будет, если использовать менее сложные деревья, установив меньшую глубину? Или что произойдет при включении в модель меньшего количества деревьев решений? Проверить эти возможности довольно просто, и мы снова видим, что алгоритм XGBoost стремится сохранить свои преимущества.

Например, если мы допустим одинаковое количество деревьев решений в ансамбле, но понизим сложность за счет уменьшения глубины деревьев, то увидим, что модель с градиентным бустингом оказывается более точной, чем модель случайного леса.

## Python

>>>## Тестирование того же количества менее сложных деревьев (10)

>>>## (max\_depth = 2)

>>>## XGBoost

>>>xgb\_clf = xgb.XGBClassifier (n\_estimetors = 10,

max\_depth = 2,

random\_state = 21)

>>>xgb\_clf. fit (X\_train, y\_train)

>>>xgb\_clf. score (X\_test, y\_test)

0.616

>>>## Случайный лес

>>>rf\_clf = RandomForestClassifier(n\_estimators = 10,

max\_depth = 2,

random\_state = 21)

>>>rf\_clf. fit (X\_train, y\_train)

>>>rf\_clf.score(X\_test, y\_test)

0.544

Это утверждение оказывается верным даже при еще большем уменьшении сложности дерева.

>>>## Тестирование того же количества менее сложных деревьев (10)

>>>## (max\_depth = 1)

>>>## XGBoost

>>>xgb\_clf = xgb.XGBClassifier(n\_estimators = 10,

max\_depth = 1,

random\_state = 21)

>>>xgb\_clf. fit (X\_train, y\_train)

>>>xgb\_clf. score (X\_test, y\_test)

0.632

>>>## Случайный лес

>>>rf\_clf = RandomForestClassifier(n\_estimators = 10,

max\_depth = 1,

random\_state = 21)

>>>rf\_clf. fit (X\_train, y\_train)

>>>rf\_clf. score (X\_test, y\_test)

0.376

Есть несколько возможных причин, объясняющих эффективность и предполагаемое преимущество градиентного бустинга деревьев над случайными лесами. Важно помнить, что нам наверняка не известно, насколько полезными окажутся признаки, избранные для классификации. Этот пример показывает, в каких ситуациях бустинг (градиентный бустинг дерева) оказывается предпочтительнее баггинга (случайного леса). Бустинг с большей вероятностью игнорирует бесполезные признаки, поскольку всегда учитывает полный набор признаков и выделяет наиболее релевантные из них, в то время как отдельные деревья, возникающие при баггинге, основываются на менее значимых признаках.

Этот пример также указывает на то, что наибольшая эффективность бустинга достигается при работе с признаками, сгенерированными специальными программными пакетами, описанными в предыдущей разделе (7). Не боясь создания для временных рядов нескольких сотен признаков — гораздо большего количества, чем можно осмыслить, — смело обращайтесь к бустингу, позволяющему избежать действительно катастрофических результатов.

Градиентный бустинг дерева эффективнее всего работает с большими наборами данных, включая длинные временные ряды. Точность прогнозирования и скорость обучения напрямую зависят от избранного способа реализации. Помимо XGBoost, обязательно обратите внимание на библиотеки LightGBM и CatBoost. Как известно, последние пакеты работают заметно быстрее, чем XGBoost, хотя и с ощутимо меньшей точностью вневыборочного тестирования.

***Классификация и регрессия***

В предыдущих примерах рассмотрены принципы классификации временных рядов с использованием алгоритмов случайных лесов и градиентного бустинга. Не менее эффективно они применяются для прогнозирования временных рядов. Многие специалисты по статистике считают, что в прогнозировании данных машинное обучение позволяет добиться не намного больших, если не меньших, успехов, чем традиционный статистический подход в анализе временных рядов.

Тем не менее за последние несколько лет градиентный бустинг деревьев стал опорным камнем технологий прогнозирования, часто превосходя традиционные статистические модели особенно при обработке больших наборов данных — как в соревнованиях по прогнозированию, так и в промышленных приложениях. Тем не менее при его использовании приходится уделять много внимания настройке параметров моделей, а также подготовке признаков временных рядов.

Одно из неоспоримых преимуществ моделей градиентного бустинга заключается в способности автоматического отсеивания нерелевантных или зашумленных признаков и выделении наиболее важных из них. Однако одного только этого недостаточно для получения самых передовых по характеристикам моделей. Даже для, казалось бы, автоматического метода, такого как градиентный бустинг деревьев, выходные данные могут оказаться не намного лучшими по качеству входных данных На этом фоне самым важным способом улучшения модели по-прежнему будет предоставление качественных и хорошо протестированных признаков для входных данных.

Существует много вариантов улучшения текущей модели. В частности, можно обратиться к специальным функциям пакета XGBoost для определения степеней важности признаков. Это поможет разделить полезные и неиспользуемые признаки и впоследствии расширить набор данных, добавив первые к тем, которые были признаны полезными ранее. Кроме того, можно выполнить поиск по сетке гиперпараметров, чтобы более точно настроить параметры модели. Наконец, всегда можно более тщательно изучить необработанные временные ряды размеченных данных, попытавшись распознать в них ошибочные данные, которые невозможно представить текущим набором признаков. Стоит задуматься о добавлении признаков, которые наилучшим образом описывают ошибочно распознанные данные, что приведет к дополнительному их уточнению.

***Кластеризация***

Главная идея кластеризации состоит в отнесении точек данных с похожими характеристиками к разным группам, представляющим отдельные цели для анализа. Такой подход оказывается верным не только для наборов временных рядов, но и для любых других видов данных.

Как и в любом другом случае, я исхожу из предположения, что вы немного знакомы с соответствующими концепциями машинного обучения — хотя бы применимо к данным, не имеющим выраженной временной направленности. Если же вы не знакомы с кластеризацией, то для начала хотя бы кратко ознакомьтесь с ее основными принципами (https: //perma.cc/36EX-3QJU).

В анализе временных рядов кластеризация может применяться как для классификации, так и для прогнозирования. В первом случае алгоритмы кластеризации применяются для определения требуемого количества кластеров на этапе обучения. Затем эти кластеры используются для выделения типов временных рядов и отнесения целевых выборок к соответствующей группе.

При прогнозировании кластеризация задействуется в явном виде или проявляется в опосредованной форме — для определения метрик расстояний (подробнее о них вы узнаете далее). Существует несколько вариантов построения прогнозов на будущем горизонте й, основанных на кластеризации данных и связанных с ними методов. Помните, что в подобном случае наблюдается не полный временной ряд, а только его первые *N* шагов, по которым предсказываются значения на временном шаге *N* + *h*. Здесь также возможно несколько вариантов действий.

Один из них заключается в составлении прогноза по типичному поведению данных, характерному для рассматриваемого класса. Сначала на основе первых *N* временных шагов определяется кластер, к которому относится выборка временного ряда, а затем прогнозируется поведение по принадлежности к такому кластеру.

В частности, на этом этапе нужно отследить изменения в значениях временных рядов между временными шагами *N* и *N* + *h*. Обратите внимание, что первоначальная кластеризация на основе их первых *N* шагов выполняется для всех временных рядов, а не для всех частей временного ряда, чтобы избежать упреждения.

Еще один вариант — предсказать будущее поведение выборочного временного ряда по поведению его ближайшего соседа (или соседей) в выборочном пространстве. В этом сценарии находятся ближайшие соседи выборки временного ряда, для которых известна полная траектория, на основе показателей из первых *N* временных шагов, а затем усредняется поведение таких ближайших соседей в момент *N* + *h*, чтобы получить прогноз для текущей выборки.

В случае классификации и прогнозирования наиболее важным фактором выступает способ оценки сходства между временными рядами. Кластеризация может быть выполнена для различных расстояний, и способам их определения в задачах высокой размерности посвящено большое количество исследований. Например, каково расстояние между двумя соискателями? Каким будет правильное расстояние между двумя выборками показателей крови? Такие задачи давно изучаются в статическом анализе данных — нам предстоит разобраться в них в контексте анализа временных рядов.

В нашем распоряжении есть два больших класса метрик расстояния (схожести), применяемых в методах кластеризации по отношению к данным временных рядов.

Расстояние, основанное на признаках Создайте признаки для временных рядов и обработайте их как координаты, для которых необходимо рассчитать данные. Это не полностью решает проблему выбора метрики расстояния, но сводит ее к стандартной задаче определения расстояния в наборе статических данных.

Расстояние на основе необработанных данных временного ряда Найдите способ определения “близости” временных рядов, предпочтительно таким образом, чтобы учитывать разные временные масштабы, различное количество измерений и другие вероятные отличия в выборках временных рядов.

Далее мы будем применять оба способа определения метрики расстояния к набору данных временного ряда, в котором каждая выборка представляет проекцию рукописного слова из двумерного изображения в одномерный временной ряд.

***Генерация признаков из данных***

Ранее мы обсудили способы создания и выбора признаков. Далее нам предстоит оценить расстояние между наборами данных временных рядов на основе сходства их признаков.

В идеальном сценарии, используя дерево для оценки важности признаков, из набора данных отсеиваются малозначимые и неинтересные временные ряды.

При этом такие признаки не применяются в методиках расчета расстояния, поскольку могут ошибочно указывать на различия между двумя временными рядами, в то время как самом деле они не свидетельствуют о сходстве целевых классов в задаче классификации или результатах прогнозирования.

Как и в случае с набором данных ЭЭГ, анализ данных стоит начать с изучения отдельных примеров классов и определения очевидных различий во временной структуре наборов данных временных рядов.

В качестве исходных данных будем рассматривать подмножество набора FiftyWords (https://oreil.ly/yadNp), доступного для загрузки из репозитория UEA and UCR Time Series Classification. Этот набор данных был создан авторами статьи о кластеризации рукописных слов в исторических документах (https://oreil.ly/01UJ8) еще в 2003 году. В этой статье авторы предложили использовать своеобразные “профили слов” для масштабирования двумерного изображения рукописного слова в одномерную кривую, представленную одинаковым количеством измерений независимо от длины слова. Набор данных в репозитории не совпадает с описанным в документе, но имеет схожую структурную организацию. Целью оригинальной работы было разработка метода унифицированного обозначения всех схожих или идентичных слов в документе, что позволяет в дальнейшем обозначить их цифровой подписью (подвиг, который в наши дни “элементарно” осуществляется нейронной сетью, прошедшей 20-летний путь постоянного усовершенствования).

Таким образом, в этом примере анализ данных основан на работе с профилями проекции слову в которых термин “проекция” указывает на преобразование изображения из двумерного вида в одномерный, и это очень важный этап, поскольку размерность данных имеет первостепенную важность в анализе временных рядов. Обратите внимание, что ось времени не содержит временных меток, а представляет последовательность написанных слов слева направо. Тем не менее структура таких данных полностью повторяет принятую для временных рядов (упорядоченные и равномерно распределенные значения), поэтому в дальнейшем они будут описываться терминами время и временной, хотя это и не совсем верно. В рассматриваемом сценарии это не имеет принципиального значения.

Примеры представления разных слов приведены на рисунке 8.4 (Обратите внимание, что информация о фактическом содержании каждого “слова” остается недоступной и не особенно актуальна при сборе всего исходного набора данных. Идея заключается в том, чтобы обозначить одинаковые слова одной и той же подписью, чтобы в дальнейшем сократить усилия по разметке документов.)

Рис. 8.4. Профили проекции трех разных слов (12,23 и 9) сильно различаются между собой. На них просматриваются признаки, по которым различаются слова: временное расположение (ось х) самого большого или второго по величине пика, количество локальных пиков, общий диапазон значений и средняя выпуклость кривых

Рассматривая полученные данные, и особенно анализируя очевидные закономерности, проявляющиеся при визуальном изучении графиков, так же, как в данных ЭЭГ, можно определиться с важными для дальнейшего анализа признаками, например, указывающими высоту и расположение пиков, а также с отдельными их характеристиками, в частности с крутизной и формой вершины.

Многие из таких признаков становятся более значимыми при распознавании изображений, чем временных рядов, и могут использоваться в качестве основы для генерации признаков в других задачах. Зачастую визуализация данных, выполняемая очень просто, может предоставить намного больше понятной и полезной информации, чем утомительное изучение длинных последовательностей значений временных рядов. Таким образом, определяясь с признаками, не пренебрегайте изучением графического представления данных. Один только этот факт указывает на сложность и важность операции генерации признаков в анализе временных рядов. В некоторых случаях различия в классах временных рядов проявляются достаточно сильно, но описать их в коде оказывается далеко не так просто, как хотелось бы. Или вдруг выясняется, что написание такого кода оказывается чрезвычайно затратным занятием.

В последнее десятилетие самым перспективным инструментом классификации изображений стало глубокое обучение. При наличии достаточного количества данных можно обучить классификатор глубокого обучения на изображениях наших графиков. Но на данный момент мы не будем акцентировать свое внимание на нем, а обсудим сложности более простых вариантов реализации. В частности, генерировать признаки, определяющие местоположение каждого пика, достаточно сложно, поскольку обнаружение пиков требует значительных программных и вычислительных ресурсов и больше сродни искусству.

Воспользуемся для визуальной оценки данных одномерной гистограммой, построенной либо для всех, либо только для отдельных примеров классов. Такой подход оказывается менее затратным в вычислительном плане при нахождении пиков или поиске других критериев, представляющих целевые формы, которые проявляются во временных рядах. В следующем примере на графике отображаются те же члены класса, которые были представлены на предыдущем графике, но теперь в сопровождении одномерных гистограмм (рис. 9.5).

## Python

pit. subplot (3, 2, 1)

pit .plot (words. iloc [1,1: -1]

pit. title ("Word = " + str(words.word[1]), fontweight = 'bold’)

pit. subplot (3, 2, 2)

pit .hist (words. iloc [1, 1:-1], 10)

pit. subplot (3, 2, 3)

pit .plot (words. iloc [3, 1:-1])

pit .title ("Word = " + str(words.word[3]), fontweight = 'bold')

pit. subplot (3, 2, 4)

pit .hist (words. iloc [3, 1:-1], 10)

pit. subplot (3, 2, 5)

pit .plot (words. iloc [5, 1:-1])

pit.title ("Word = " + str(words.word[11]), fontweight = 'bold')

pit. subplot (3, 2, 6)

pit .hist (words. iloc [5, 1:-1], 10)

Нужно убедиться, что рассматриваемые примеры не являются выбросами в сравнении с другими экземплярами этих же слов. По этой причине построим двумерную гистограмму представления двух слов, чтобы получить представление об их отдельных вариантах (рисунок 9.6).

## Python

х = np.array ([])

у = пр.array ([])

w = 12

selected\_words = words[words.word == w]

selected\_words. shape

for idx, row in selected\_words.iterrows():

у = np.hstack ([у, row[l:271]])

x = np.hstack ([x, np. array (range (270) )])

fig, ax = pit. subplots ()



Pиc. 8.5. Еще один способ количественного описания классов при поиске полезных признаков. В частности, гистограмма отдельных экземпляров классов показывает, что такие характеристики гистограммы, как количество локальных пиков, асимметрия и эксцесс, могут оказаться хорошей и полезной заменой отдельных атрибутов кривых временных рядов, будучи простыми для визуального анализа, но чрезвычайно сложными в программной реализации

Рис.8.6. Двумерная гистограмма одномерных проекций слова 12. На оси у откладываются значения на отдельных временных шагах, а ось х включает 270 временных шагов для каждой выборки временного ряда/проекции слова

На рисунке 8.6 показана двумерная гистограмма для всех экземпляров слова 12 в наборе данных. Хотя отдельная кривая на рисунке показывает, что нужно сосредоточиться на поиске двух больших пиков, которые доминируют во временном ряду, гистограмма показывает, что, вероятнее всего, общим для большинства членов этого класса будет плоский участок между пиками, расположенными между временными шагами от 120 до 200.

Двумерную гистограмму также можно использовать для определения точки отсечения для максимального пика в заданном классе, который, по-видимому, находится между временными шагами 50 и 150, — всегда можно написать функцию, которая ответит на вопрос «достигается ли максимальное значение между точками 50 и 150?»

Рис. 8.7. Двумерная гистограмма одномерных проекций слова 23. На оси у откладываются значения на отдельных временных шагах, а ось х включает 270 временных шагов для каждой выборки временного ряда/проекции слова

Подобным образом построим еще одну двумерную гистограмму — на этот раз для слова 23, профиль которого характеризуется большим количеством маленьких выпуклостей (см. пример на рисунке 8.5), признаки которых также сложно поддаются программному вычислению (рисунок 8.7).

Неудивительно наблюдать сильно “размазанную” гистограмму для слова 23 (рисунок 8.7), учитывая тот факт, что даже в примере, представленном графиками на рисунке 8.5, проявляется достаточно много признаков, чтобы ожидать сильного размазывания для выборок с неточно совпадающими признаками. Тем не менее мы также видим, что максимальное значение этого класса относится к непересекающемуся диапазону временных шагов, в отличие от слова 12. В этом классе максимальное значение наблюдается после временного момента 150, что вполне проецируются в другой тип одномерных данных, по которым и определяются признаки). Таким образом, учитываются большие “мазки”, проявляющиеся на двумерных гистограммах и указывающих на наличие пиков, положение которых не отличается особой устойчивостью. Генерация второй характеристической формы для каждой проекции слова по гистограмме, а не исходной проекции слова, может оказаться более надежным способом. Гистограммы показывают, какие виды значений появляются во временном ряду, но не характеризуют их точное расположение, что крайне важно для нашего анализа, учитывая, что пики в проекциях не имеют стабильных временных мест расположения.

Сначала сгенерируем признаки для временного ряда длиной 270 временных точек. В нашем случае функция генерации признаков имеет сокращенное название, что сделано исключительно в целях улучшения читабельности кода.

from cesium import featurize.featurize\_time as ft

## Python

word\_vals = words.iloc[:, 1:271]

times = []

word\_values = []

for idx, row in word\_vals. iterrows () :

word\_values.append(row.values)

times.append(np.array ([i for i in

range(row.values.shape[0])]))

features\_to\_use = ['amplitude',

' percent\_beyond\_l\_s td',

' percent\_close\_to\_median' ]

featurized\_words = ft (times = times,

values = word\_values,

errors = None,

features\_to\_use = features\_to\_use,

scheduler = None)

Теперь построим гистограммы и используем их в качестве временного ряда, для которого генерируются признаки. (При построении гистограмм мы будем придерживаться того же принципа, что и в случае визуализации проекций слов, в которых на оси *х* откладывается не время, а другая равномерно и последовательно упорядоченная величина, которая, тем не менее, рассматривается как время. В нашем примере при генерировании признаков будем рассматривать ось *х* как временную.)

## Python

## Генерирование признаков по гистограммам

times = []

hist values = []

for idx, row in words features.iterrows():

hist values.append(np.histogram(row.values,

bins=10,

range=(-2.5, 5.0))[0] + .0001)

## обработка нулей вызывает

## дополнительные сложности

times. append (np.array([i for i in range(9)]))

features to use = ["amplitude",

"percent\_close\_to\_median",

"skew"]

featurized hists = ft(times = times,

values = hist values,

errors = None,

features to use = features to use,

scheduler = None)

Легко удостовериться, что во всех гистограммах учитывается одинаковое количество интервалов, относящихся к общему диапазону значений, представляемому параметрами, которые передаются функции np.histogram(). Обеспечив общность диапазона значений, образующих единую “временную” ось, гистограммы можно сравнивать непосредственно при генерации признаков временных рядов. Если отказаться от проведения такого согласования, то сгенерированные признаки не обязательно будут релевантными при сравнении одной гистограммы с другой.

Наконец, объединим оба источника признаков.

## Python

features = pd.concat([featurized\_words.reset\_index(drop=True),

featurized\_hists],

axis=l)

***Временные метрики расстояния***

При проведении кластерного анализа нужно определиться с метриками расстояния. Ориентируясь по признакам временных рядов, как показано выше, можно придерживаться самых разных стандартных методик, что характерно для кластерного анализа большинства перекрестных данных. Если вы не знакомы с принципами выбора метрик расстояния в подобных случаях, то обязательно обратитесь к соответствующим справочным руководствам (https://perma.сс/MHL9-2Y8A).

Сосредоточимся на задаче определения сходства временных рядов по метрикам расстояний. Одна из наиболее часто применяемых для таких целей метрик называется динамической деформацией времени (Dynamic Time Warping — DTW). Механизм DTW прекрасно подходит для кластеризации временных рядов, наиболее характерной особенностью которых является их общая форма, как в случае рассмотренной ранее визуализации проекций слов.

Название метода основано на методологии, которая базируется на временной “деформации” рядов данных для выравнивания вдоль временной оси с целью последующего сравнения форм. Концепцию динамического искажения времени лучше всего проиллюстрировать графически (рисунок 8.8). Временная ось (х) деформируется, т.е. расширяется или сжимается так, чтобы обеспечить наилучшее выравнивание точек между двумя кривыми (т.е. двумя временными рядами) для последующего сравнения их форм.

Рис. 8.8. Механизм динамической деформации времени. Каждая точка одного временного ряда сопоставляется с точкой альтернативного временного ряда. При этом однозначного сопоставления точек не требуется, что имеет несколько важных последствий: (1) временные ряды не обязательно должны иметь одинаковую длину или даже масштабы времени, главное — их форма; (2) время не всегда движется вперед в процессе подгонки и может протекать с разной скоростью в каждом временном ряду. Под движением времени подразумевается продвижение по кривой в направлении оси х

Заметьте, что при стандартной параметризации алгоритма фактические значения на временной оси одной кривой совершенно не совпадают со значениями другой временной оси. Никто не запрещает сравнивать временной ряд с наносекундным интервалом с другим временным рядом, данные которого масштабированы для нескольких тысячелетий (скорее всего, это будет далеко не самым разумным решением). Цель этого алгоритма — сравнить одни только визуальные формы без учета временной протяженности данных. Действительно, термин “время” в данном случае рассматривается в общем концептуальном понимании как направление равномерного распределения набора точек вдоль оси х, а не в строгом физическом проявлении.

Перечислим правила работы DTW.

• Каждая точка одного временного ряда должна соответствовать как минимум одной точке другого временного ряда.

• Первый и последний индексы каждого временного ряда должны сопоставляться с таковыми в другом временном ряду.

• Масштабирование данных должно выполняться так, чтобы обеспечить прямой (вперед), а не обратный порядок движения времени. Обратного хода времени можно избежать, если исключить сопоставление точек одного временного ряда с точками другого временного ряда, которые уже были пройдены на временной оси. Однако время не обязано должно двигаться вперед равномерно. Например, два последовательных временных шага в исходном ряду могут быть деформированы (сжаты во времени) таким образом, чтобы подогнать соответствующие значения к одному и тому же месту на оси х (см. первый излом верхней кривой/сплошной линии на рисунке).

Указанные правила предопределяют много разных способов временного выравнивания — выбранный нами вариант обеспечивает минимизацию расстояния между кривыми. Это расстояние, или функция стоимости, часто измеряется как сумма абсолютных разностей между совмещаемыми точками, где под абсолютной разностью подразумевают разницу между значениями точек.

Получив качественное представление о работе алгоритма DTW, давайте познакомимся с примером его программной реализации.

## Python

def distDTW(tsl, ts2) :

## Настройка

DTW={}

for i in range(len(tsl)) :

DTW[ (i, -1) ] = np.inf

for i in range(len(ts2)) :

DTW[ (-1, i) ] = np.inf

DTW[ (-1, -1) ] =0

## Пошаговый расчет оптимальных значений

for i in range(len(tsl) ) :

for j in range(len(ts2) ) :

dist = (tsl [i] - ts2[j])\*\*2

DTW[(i, j)] = dist + min (DTW [(I - 1, j)],

DTW[(i, j - 1)],

DTW [ (i-1, j -1)])

## Это пример динамического программирования

## Рассчитав полный путь, можно получить

## соответствующее расстояние

return sqrt(DTW[len(ts1) - 1, len(ts2) - 1])

Как указано в комментариях, эта задача решается методами динамического программирования, а вычисление расстояния DTW является классической задачей динамического программирования. Мы можем пошагово отследить пути от начала каждого временного ряда до конца, и построить решение, основываясь на знаниях о предыдущих шагах при принятии последующих решений.

Существует целый ряд реализаций DTW с различными принципами повышения эффективности поиска оптимального или почти оптимального решения. Им следует уделить особо пристальное внимание, особенно при работе с большими наборами данных. Есть и другие способы измерения расстояний между временными рядами.

Ниже описаны некоторые из них.

*Расстояние Фреше*

Это минимальное расстояние между двумя кривыми, обеспечивающее беспрепятственное прохождении по ним при временной деформации. Эта метрика расстояния часто объясняется на примере прогулки человека и его собаки, которые движутся по двум разным траекториям, но постоянно связаны между собой поводком. Каждому из них необходимо пройти свою кривую от начала до конца, они могут двигаться с разными скоростями и изменять их в произвольных точках кривой при перемещении в одном и том же направлении. Расстояние Фреше — это самая короткая длина поводка, при которой можно пройти путь по оптимальным траекториям (при условии, что человек и его собака смогут их найти!).

*Корреляция Пирсона*

Измерять расстояния между двумя временными рядами можно по корреляции между ними. В отличие от других способов здесь расстояние между временными рядами минимизируется в результате максимизации корреляции между ними. Это сравнительно простой в вычислительном плане метод. Однако он требует использования во временных рядах одинакового количества точек данных или понижения частоты дискретизации одной из выборок до таковой в другой выборке. Временная сложность вычисления корреляции составляет О(*n*), что указывает на высокую эффективность метода с точки зрения вычислительных затрат.

*Самая длинная общая подпоследовательность*

Эта мера расстояния подходит для временных рядов, которые представляют собой последовательность категориальных или целочисленных значений. В таких случаях при рассмотрении сходства двух временных рядов можно определить длину самой длинной общей подпоследовательности, т.е. наибольшую длину последовательности значений, которые точно совпадают, хотя их точное расположение во временном ряду не требуется для совпадения. Как и в случае с DTW, такой подход предполагает нахождение схожих форм, а не точное совпадение временных интервалов, в которых они проявляются. Также отметим, что, как и DTW, но в отличие от корреляции Пирсона, в этом методе не требуется, чтобы временные ряды имели одинаковую длину. Родственная мера — это расстояние редактирования, т.е. количество изменений, которые необходимо внести в один временной ряд, чтобы сделать его идентичным другому.

***Расстояние или сходство***

В источниках, посвященных поиску метрик расстояний между временными рядами, для описания упомянутых выше показателей часто применяется термин “сходство” В большинстве случаев эти термины рассматриваются как взаимозаменяемые — описывающие степень подобности временных рядов между собой. Тем не менее отдельные показатели могут показаться более подходящими, например расстояние Фреше, которое вычисляется в соответствующих единицах измерения (“футы” или “кг/доллар” или любых других единицах представления данных временного ряда). Остальные метрики расстояния, в частности корреляция, единицами измерения не снабжаются.

Иногда найти простое, но удачное решение помогает творческий подход. Всегда старайтесь как можно более точно оценить стоящие перед вами цели. Обратимся к публикации на сайте Stackoverflow (<https://perma.cc/389W-68AH>), в которой рассматривается задача поиска расстояния в конкретном приложении, а именно — при классификации временных рядов для дальнейшего сопоставления найденных классов с одним из трех центроидов, полученных в предыдущем кластерном анализе. Рассматриваются три следующих класса.

• Прямая линия.

• Пик в начале временного ряда, а затем — прямая линия.

• Пик в конце временного ряда, а затем — прямая линия. Автор публикации обнаружил, что несколько стандартных метрик расстояний, включая эвклидово и DTW, не подходят для решения задачи. Метрика DTW дает слишком грубую оценку, рассматривая любые временные ряды с пиком подобно временным рядам с пиками в конце и начале ряда (именно поэтому DTW нельзя считать панацеей, даже несмотря на высокую вычислительную затратность метода!).

Один из дотошных комментаторов предложил провести преобразование, которое повысило бы точность вычисления расстояний, а именно: сравнивать не исходные временные ряды, а временные ряды накопленных сумм. В результате как эвклидовы метрики расстояния, так и DTW приводят к правильному упорядочению — временной ряд с пиком в начале показал наименьшее расстояние для прототипа своего класса, но не равное расстоянию для прототипа с пиками в начале и конце временного ряда. Это напоминает подход, принятый в одном из более ранних примеров анализа, в котором для принятия модели ARIMA достаточно правильно преобразовать временной ряд, необработанные данные которого не удовлетворяют всем необходимым условиям.

**Избегайте эвклидовых метрик расстояния при сравнении временных рядов** Возможно, вы заметили, что эвклидово расстояние не упоминалось в приведенном выше обсуждении метрик подобия временных рядов. Эвклидово расстояние, как правило, плохо подходит для оценки сходства временных рядов. Чтобы понять, почему, сравним две синусоиды и прямую линию. Если синусоиды имеют одинаковый период, но разные фазы (смещены по оси времени х) или характеризуются разными амплитудами, то прямая линия будет иметь меньшее эвклидово расстояние до одной из синусоид, чем до другой. Это далеко не тот результат, который ожидалось получить, но он указывает на отдельные преимущества рассмотренных ранее метрик.

• Компенсация фазы — смещение по временной оси, которое не играет большой роли при сравнении.

• Распознавание сходства форм, а не сходства величин.

Ниже представлено простое упражнение обязательное для реализации вами.

Все же, остановив свой выбор на эвклидовом расстоянии, рассмотрите возможность использования его совместно с преобразованием Фурье. В этом случае уменьшение количества измерений осуществляется за счет ограничения частоты, что подходит для большинства данных временных рядов, поскольку высокие частоты оказывают слабое влияние на общую форму и динамику временного ряда. Другой вариант заключается в аппроксимации по методу символьной агрегации (SAX — Symbolic Aggregate Approximation), разработанному в 2007 году для уменьшения размерности временных рядов перед вычислением нижней границы эвклидова расстояния.

К сожалению, не существует универсального правила для выбора метрики расстояния. Вам придется проявить недюжинные старания, чтобы найти сбалансированное решение, учитывающее следующие факторы.

• Минимизация использования вычислительных ресурсов.

• Выбор расстояния, подчеркивающего признаки временного ряда, которые лучше всего соответствуют конечной цели.

• Метрика расстояния должна учитывать и подчеркивать сильные/слабые стороны аналитических методов, используемых для сравнения временных рядов. Например, кластеризация по методу /с-средних не предполагает попарного сравнения расстояний, но сводит к минимуму отклонения, так что в этом методе имеет смысл обратиться к эвклидовым метрикам.

**Код кластеризации**

Теперь, когда мы обсудили, как генерируются признаки для кластерного анализа и рассмотрели принципы измерения расстояние между временными рядами, выполним кластеризацию по выбранным признакам и согласно матрице попарных расстояний DTW с последующим сравнением результатов.

***Иерархическая кластеризация нормализованных признаков***

Вычислим признаки для слов, представленных в виде временных рядов, как для исходных записей, так и для их гистограмм. В дальнейшем такие признаки будут использоваться в совершенно разных масштабах, поэтому для применения к ним единой метрики расстояния их необходимо нормализовать, что является стандартной процедурой в задачах кластеризации, основанной на признаках.

## Python

from skleam import preprocessing

feature\_values = preprocessing.scale(features.values)

Выберем алгоритм иерархической кластеризации и проведем обучение для 50 кластеров, поскольку в дальнейшем будем сопоставлять эти кластеры с 50 словами в исходном наборе данных.

## Python

from sklearn. cluster import AgglomerativeClustering

feature\_clustering = AgglomerativeClustering(n\_clusters = 50,

linkage = 'ward')

feature\_clustering.fit(feature\_values)

words ['feature\_labels' ] = feature\_clustering. fit\_predict (p)

Давайте посмотрим, наблюдаются ли соответствия в подписях слов и кластеров (последние создаются независимо от оригинальных подписей слов).

## Python

from sklearn. me tries. cluster import homogeneityscore

homogeneity\_score(words.word, words.feature\_labels)

Нам повезло работать с размеченными данными; в противном случае существует вероятность сделать неправильные выводы по данным полученных кластеров.

В нашем случае около половины кластеров тесно связаны с одним словом. Вернувшись назад и попробовав улучшить результат, можно прийти к следующим выводам.

• Использовано всего шесть признаков. Это очень мало, поэтому их количество лучше увеличить.

• Нужно обратиться к признакам с относительно слабой корреляцией, что в нашем случае не было сделано.

• Полезных признаков явно недостаточно. При визуальном изучении данных были подмечены признаки, которые не участвовали в анализе, например количество и расположение характерных пиков. Вероятно, нужно пересмотреть отношение к этим характеристикам, выделив их в признаки в явном или косвенном виде.

• Нужно обратиться к другим метрикам расстояний, возможно к таким, в которых отдельные признаки имеют больший вес, чем остальные, отдавая предпочтение тем признакам, которые визуально кажутся более полезными.

***Иерархическая кластеризация с помощью матрицы расстояний DTW***

Мы успешно завершили самую сложную часть метода прямой кластеризации временных рядов, основанного на вычислении матрицы попарных расстояний для DTW. Это очень затратная операция с точки зрения вычислительных ресурсов, поэтому полученные результаты лучше сохранить, чтобы иметь возможность вернуться к их анализу в случае любой необходимости.

## Python

p = pairwise\_distances(X, metric = distDTW)

## Вычисления выполняются долго, поэтому данные стоит сохранить

with open ("pairwise\_word\_distances. nру", "wb") as f:

np.save(f, p)

Располагая матрицей попарных расстояний, можно обратиться к алгоритму иерархической кластеризации.

## Python

from sklearn.cluster import AgglomerativeClustering

dtw\_clustering = AgglomerativeClustering(linkage = 'average',

n\_clusters = 50,

affinity = 'precomputed')

words ['dtw\_labels' ] = dtw\_clustering. fit\_predict (p)

Наконец, как и прежде, проверим соответствие подобранных кластеров известным подписям.

## Python

from sklearn.metrics . cluster import homogeneity score,

completeness\_score

homogeneity\_score(words.word, words.dtw\_labels)

completeness\_score(words.word, words.dtw\_labels)

Легко заметить, что кластеризация, выполняемая с помощью DTW, работает значительно лучше, чем кластеризация на основе признаков. Однако если вы запустите код вычисления расстояния DTW на своем компьютере, особенно на стандартном ноутбуке, то увидите, насколько больше времени отнимает выполнение алгоритма DTW по сравнению с вычислением признаков. Вполне вероятно, нужно заняться улучшением алгоритма кластеризации на основе признаков, поскольку кластеризация по расстоянию DTW не предполагает простых способов усовершенствования. Уточним варианты дальнейшего улучшения методик.

• Включить в рассмотрение признаки, а также расстояние DTW. Как с программной, так и с концептуальной точек зрения, совместное использование признаков и расстояния DTW является очень сложной для реализации задачей.

• Попробовать использовать другие метрики расстояний. Как обсуждалось ранее, правильный выбор расстояния зависит от исходных данных, поставленных целей и задач анализа. Таким образом, сначала нужно предельно точно определить цель анализа слов и только затем задуматься о том, действительно ли DTW представляет лучшую из доступных метрику расстояния для достижения поставленной цели.

**Дополнительные источники**

• Расстояния между временными рядами и метрики сходства

Meinard Muller, “Dynamic Time Warping,” in Information Retrieval for Music and Motion (Berlin: Springer, 2007), 69-84, https: //perma.cc/R24Q-UR84

В одной из глав этой книги содержится обширное описание алгоритма динамической деформации времени, включая обсуждение общих приближений, призванных уменьшить вычислительную сложность метода DTW.

Stephane Pelletier, “Computing the Frechet Distance Between Two Polygonal Curves,” (lecture notes, Computational Geometry, McGill University, 2002), https://perma.cc/5QER-Z89V

Набор лекционных заметок для Университета Макгилла, в которых предложено интуитивно понятное визуальное и алгоритмическое объяснение расстояния Фреше и способов его вычисления.

Pjotr Roelofsen, “Time Series Clustering,” masters thesis, Business Analytics, Vrije Universiteit Amsterdam, 2018, https://perma.cc/K8HJ-7FFE

Магистерская диссертация по кластеризации временных рядов, которая начинается с важного и очень подробного обсуждения основных методов вычисления расстояний между временными рядами, включающая информацию о сложности вычислений и полезные иллюстрации, значительно облегчающие интуитивное восприятие материала.

Joan Serra and Josep Ll. Arcos, “An Empirical Evaluation of Similarity Measures for Time Series Classification,” Knowledge-Based Systems 67 (2014): 305-14, https://perma.cc/G2J4-TNMX

В этой статье проводится эмпирический анализ точности вневыборочного тестирования моделей классификации, построенных с использованием семи различных метрик сходства временных рядов: эвклидова расстояния, коэффициентов Фурье, модели AR, DTW, расстояния редактирования, измененного во времени расстояния редактирования и минимальных затрат на устранения несходства. Авторы протестировали эти показатели на 45 общедоступных наборах данных из хранилища временных рядов UCR.

• Машинное обучение для временных рядов

Keogh Eamonn, “Introduction to Time Series Data Mining,” slideshow tutorial, n.d„ <https://perma.cc/ZM9L-NW7J>

Серия слайдов, посвященных предварительной обработке данных временного ряда, подвергаемых дальнейшему анализу с помощью алгоритмов машинного обучения, который основан на методах измерения сходства между временными рядами и определения мотивов, которые можно использовать для анализа и сравнения.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos, “The M4 Competition: Results, Findings, Conclusion and Way Forward,” International Journal of Forecasting 34, no. 4 (2018): 802-8, https : //perma. cc/42HZ-YVUU

В этой статье подводятся итоги конкурса М4, проведенного в 2018 году, в котором сравнивались различные методы прогнозирования временных рядов, включая многие ансамблевые методы, для произвольно выбранного набора из 100 тысяч временных рядов, включая данные, собранные с различными частотами (ежегодно, ежечасно и т.д.). В этом обзоре результатов конкурса авторы показывают, что как первое, так и второе места в конкурсе получены благодаря использованию комбинированных методик анализа данных, в значительной степени основанных на статистических исследованиях и алгоритмах машинного обучения. Подобные результаты указывают на важность понимания и применения как статистического анализа, так и машинного обучения в прогнозировании данных.