Модуль 3 Использование методов машинного обучения в анализе временных рядов.

# Введение:

Модуль дает представление об особенностях использования современных методов машинного обучения в анализе временных рядов. В том числе обсуждаются релевантные задачи и особенности подходов, которые связаны со спецификой изучаемой модальности данных.

Отдельно обсуждаются вопросы глубокого обучения нейронных сетей в анализе временных рядов. Отметим, что в анализе временных рядов глубокое обучение нейронных сетей далеко не всегда является необходимым. Такой подход чаще оправдан в случае больших объемов данных много-переменных временных рядов, чьи свойства не позволяют быть легко формализованными.

Упор в модуле сделан на решения дискретных задач, таких как задачи классификации и обнаружения аномалий. Однако следует отметить, что подходы могут быть с лёгкостью транслированы и на решения других задач анализа ВР.

В ходе изучения модуля будут изучены такие вопросы, как:

* Типы и примеры задач классификации ВР
* Особенности методов классификации ВР на основе сырых данных
* Особенности выбора признаков ВР и методов классификации на их основе, а также особенности использования комбинаций изученных методов.
* Особенности использования глубоких нейронных сетей в анализе ВР, в том числе рекуррентных сетей и других классов сетей.
* Типы аномалий во временных рядах и методы работы с ними.

На практических занятия будет произведено знакомство с методами настройки и использования:

* моделей классификации ВР в фреймворке SKTime
* методов работы с глубокими нейронными сетями в фреймворке TSAI
* методов обнаружения аномалий.

В результате прохождения модуля слушатель будет:

* Знать специфику, разновидности, и возможности использования методов машинного обучения в анализе временных рядов
* Уметь провести выбор методов анализа временных рядов с использованием машинного обучения для решения той или иной задачи.
* Владеть навыками использования методов машинного обучения в анализе временных рядов

# Юнит 18 - Специфика методов машинного обучения при анализе временных рядов!

Как уже говорилось ранее среди методов анализа временных рядов можно выделить:

Методы на основе моделей. Это методы, когда берется модельное описание задачи. Для оценки параметров модели находится выражение. На основе модели с известными параметрами строятся предсказания либо выбирается класс.

Например, линейный тренд на фоне белого гауссового шумов можно описать как

, где – модель временного ряда от параметра (который может быть непрерывным или дискретным); – искомые коэффициенты; - белый гауссов шум.

Из модели методом наименьших квадратов находят наклон ( и начальное значение . Будущие значения выражаются как .

Методы на основе данных. Это подход datascience. В данном подходе мы берем некоторый универсальный оценщик с настраиваемыми параметрами, например, многослойный персептрон.

Настройку параметров модели в методе на основе данных мы осуществляем по набору данных, соответствующему задаче. То есть набору в котором есть примеры входных данных и ожидаемых результатов. Если результаты в виде ответов для каждого примера – то у нас задача с учителем, если результаты в виде общего правила или в виде мнения эксперта, то задача без учителя. Набор данных по которому мы ведем настройку параметров мы называем тренировочным.

Чтобы оценить качество настройки параметров мы также должны иметь независимый набор данных – тестовый набор. Оценку качества мы производим по метрикам, например, точность как средняя модуля разности правильных и предсказанных значений.

Настройкой модели мы при этом называем процесс подбора коэффициентов модели под тренировочную выборку. То есть так, чтобы для большинства примеров получались как можно более верные ответы. Если речь идет о задаче с учителем, то сравнивать полученные и заданные ответы мы будем при помощи отдельной функции – т.н. функции потерь. Эта функция должна достичь точки, близкой к минимуму.

В свою очередь задачи анализа временных рядов без учителя заключаются в поиске закономерностей в наборе данных без меток. В таких задачах метки мы заменяем на некоторые общие правила, по которым будем анализировать данные. Например, правило, что все данные без сезонности – это сегмент 0, а данные с сезонностью – это сегмент 1.

Типичные задачи с учителем: регрессия; прогноз; классификация полных временных рядов или их сегментов.

Типичные задачи без учителя: поиск аномалий, когда о данных ни чего не известно, а также кластерный анализ или алгоритмы декомпозиции и сжатия данных.

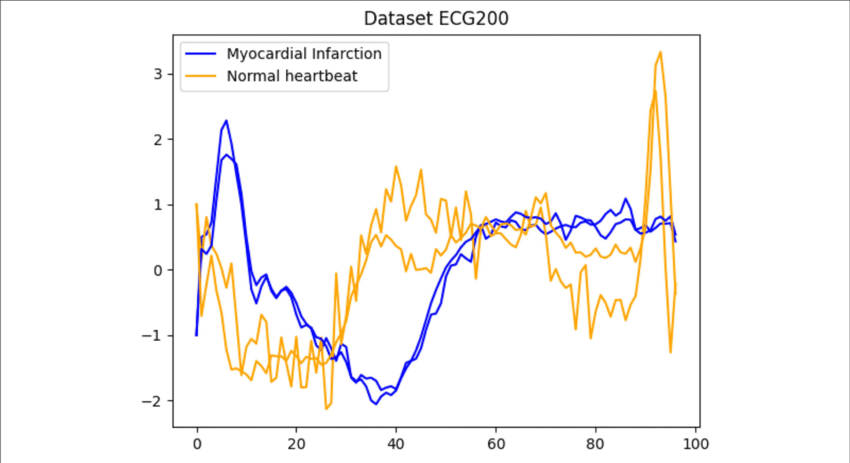
В некоторых случаях также можно выделить задачи с полуконтролируемым обучением, когда для части данных есть метки. Например, поиск аномалий в случаях, когда нам известны примеры данных без аномалий.

# Юнит 19 - Задача классификации временных рядов

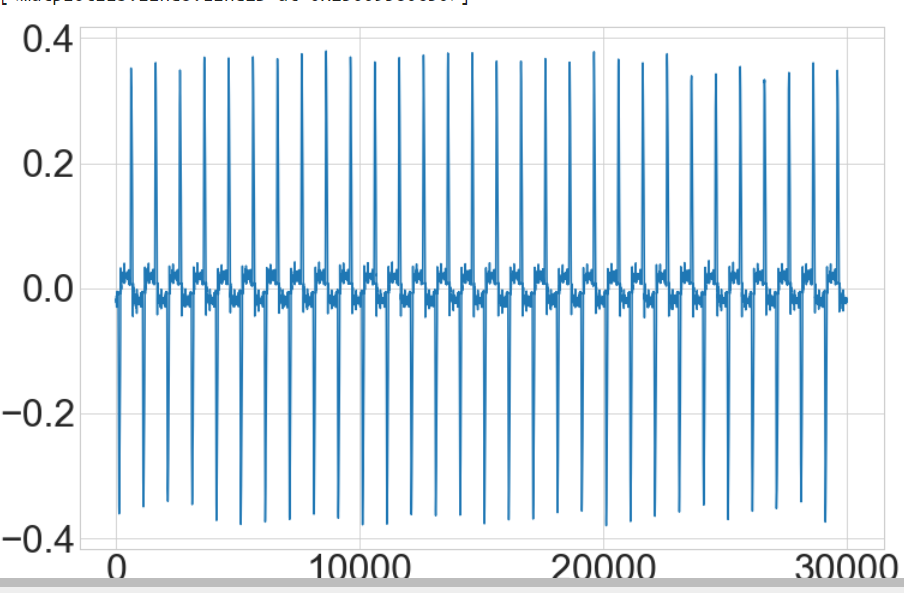
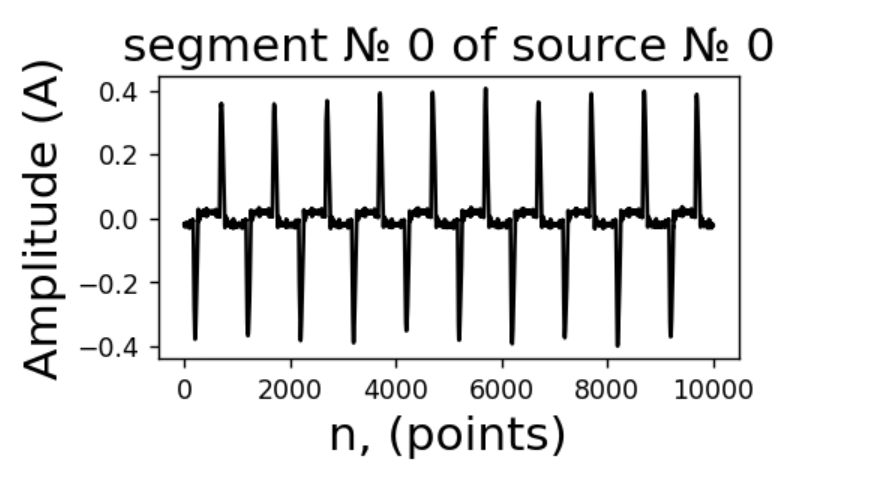
В ряде приложений анализа временных рядов возникает задача принятия некоторых дискретных решений на основе данных (то есть имеется счетное множество вариантов ответов). Если для обучения модели принятия решений имеется обучающая выборка примеров с известными ответами, то задача будет характеризована как задача классификации.

Примерами задач классификации могут стать: принятие решений о наличии заболеваний по сигналу ЭКГ; классификация особенностей речи по аудиозаписи (например, мужской или женский голос, или поиск одних и тех же фраз, записанных в разных условиях); классификация типа устройства по потребляемому току или, например, трафику в сети; а также многие другие задачи.

Как правило в данном подходе предполагается наличие определенного числа сегментов временного ряда, с наличием метки для каждого из них. При этом под словом "сегмент" понимается "кусок" временного ряда известной, ограниченной, длины. Если временной ряд достаточно большой он может быть разделен на несколько сегментов.



Пример сегментов временных рядов.

Пример длинного временного ряда, и выделенного из него сегмента.

Заметим, что выбор длины сегмента может быть как из некоторых внешних соображений, так и быть естественным (например для медицины, в силу длительности испытания). Иногда длину сегмента можно оптимизировать как некоторый гиперпараметр модели.

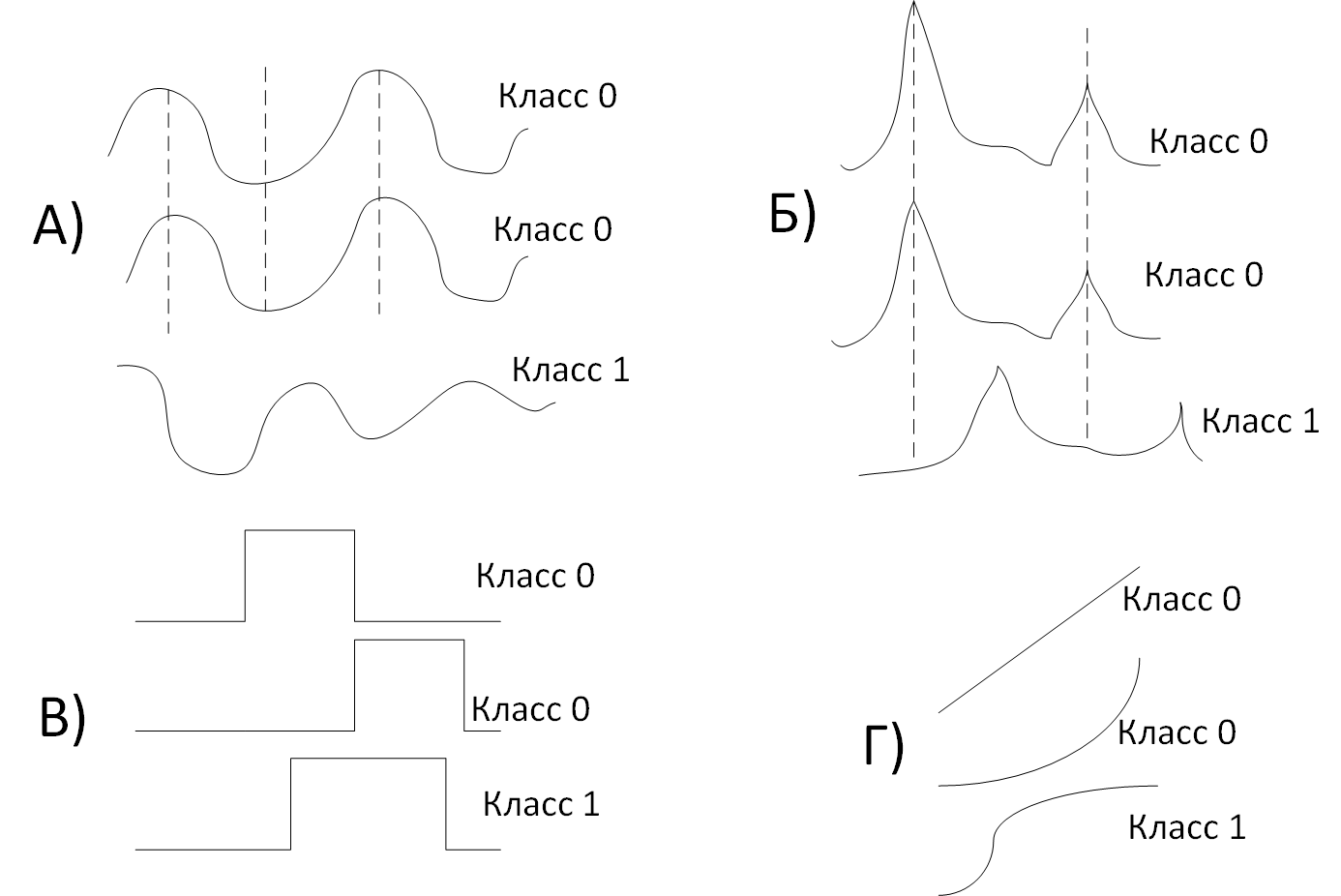
Также заметим важную особенность постановки задачи классификации для временных рядов. Очевидно, что наша задача предполагает поиск некоторой схожести для сегментов временного ряда одного класса и различия для сегментов разных классов. Но схожесть может быть разной, например могут быть следующие варианты:

схожесть во временном поведении (рисунок А);

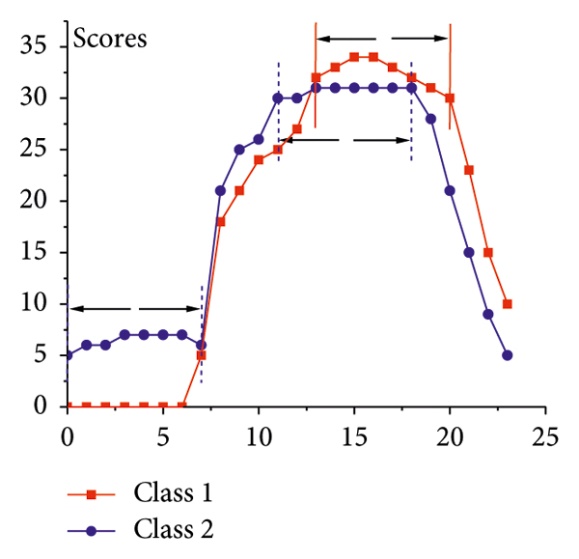
схожесть в частотном поведении (наличие определённых компонент в спектре) (рисунок Б).

схожесть по форме (патерн/шаблон, без привязки ко времени или частота повторения паттерна) (рисунок В);

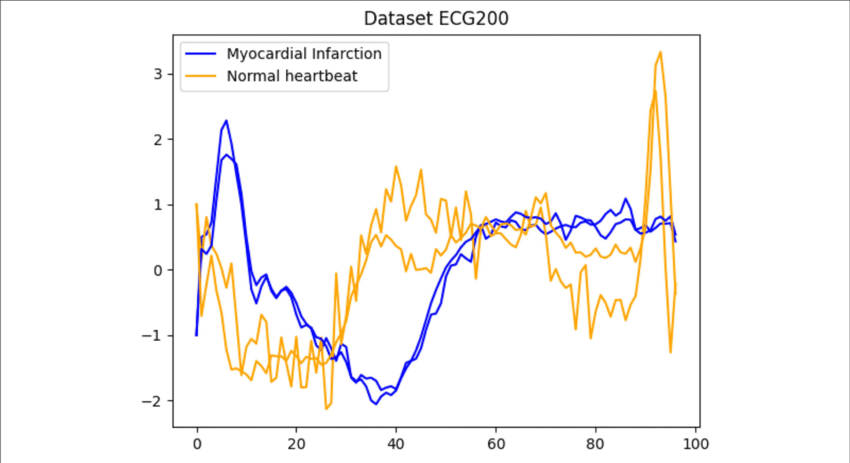
схожесть поведения компонент (монотонный тренд, характер сезонности и т.д.) (рисунок Г).



В зависимости от задачи ее можно свести к тому или иному типу задач. Например, задача определения металлических предметов на детекторе, как на рисунке, относится к классу В. Т тут нам важна форма сегмента, но не важно, где в сегменте появится это паттерн.

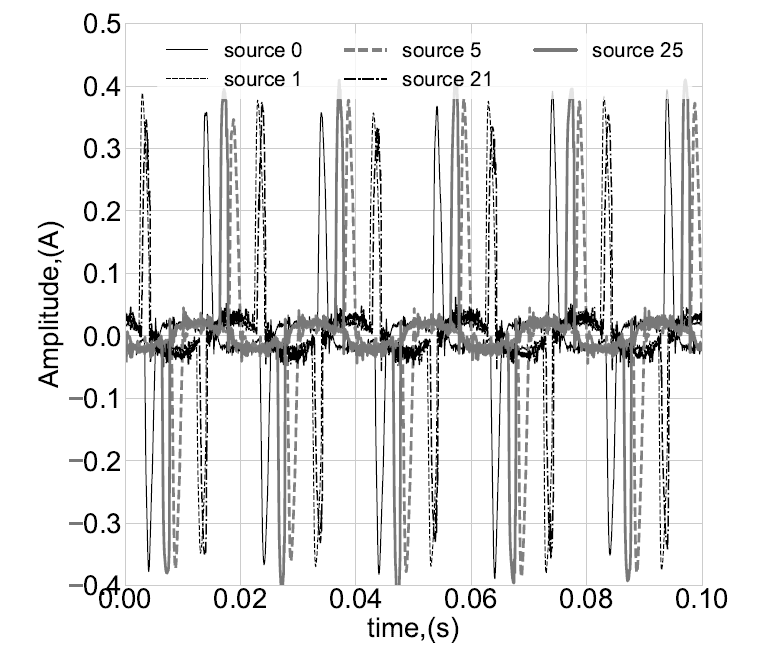


А задача диагностики по биосигналу как на рисунке ниже это задача класса А. Тут нам важно совпадение временного поведения биосигнала.

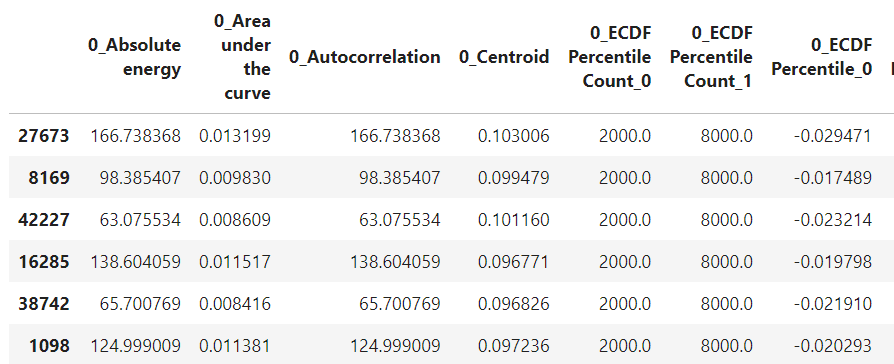


Также методы классификации временных рядов могут быть разделены на группы методов.

* Метода на основе признаков, когда из каждого сегмента мы выбираем признаки, например, максимальное либо среднее значение или период сезонности. Такие признаки, как правило, интерпретируемые, а сами методы достаточно быстры. Фактически результат выделения признаков сводится к некоторой таблице.
* Методы на основе сегментов ряда – то есть самих данных, когда учитывается вся информация из ВР. Такие методы менее интерпретируемые и как правило требуют больше вычислительных ресурсов. Однако потенциально содержат всю информацию о временных рядах. То есть могут дать точность выше, чем классификация по ряду признаков.



Пример полных сегментов для разных классов (обозначены по разному)



Пример признаков, выделенных из данных, приведённых выше. Данный пример это задача идентификации устройств по току в сети. Эта задача относится к классу Б – так как нам важно точное совпадение частотного поведения сегментов. По этой информации мы принимаем решение о идентификации устройств.

# Юнит 20 Методы классификации временных рядов

Методы на основе сегментов данных как таковых подразумевают поиск "схожести" таких участков ряда для каждого класса. При этом "схожесть" определяется как значение определенной метрики, например, расстояние.

Самый простой пример таких подходов 1-ближайший сосед (1-nearest-neighbor) c эвклидовым расстоянием. Как мы увидим позже этот пример - не всегда хороший выбор.

Достоинствами подхода на основе сегментов данных являются потенциально высокая точность. Однако, для работы подход требует достаточно много вычислительных ресурсов и не всегда позволяет "правильно" учесть всю содержащуюся в рядах информацию.

Также отметим, что результаты работы для подхода на основе признаков не всегда являются интерпретируемыми, что может представлять отдельную проблему.

В ряде случаев более удобным может является рассмотрение каждого сегмента временного ряда в виде вектора известной длины. В данном случае нет необходимости в поиске путей формализации каких-либо особенностей ВР. Вектора с соответствующими метками класса используются "на прямую". Вектора могут быть поданы на вход традиционных классификаторов, как например, kNN; SVM или на вход нейронной сети.

Как уже отмечалось наиболее простым - базовым выбором классификатора, в данном случае может быть метод 1-ближайшего соседа (1-nearest-neighbor) c эвклидовым расстоянием. Однако, эвклидово расстояние не всегда подходит. Например, эвклидово расстояние позволило бы решить только проблему А). Если сегменты имеют некоторые, даже небольшие, расхождения этот метод не подойдёт. Нужно использовать т.н. эластичные расстояния.

Эластичные меры расстояний являются модификациями традиционных подходов, устойчивыми к небольшим изменениям в данных. Среди таковых в анализе временных рядов наиболее универсальным подходом можно считать т.н. Расстояние динамической трансформации (деформации) по времени (Dynamic Time Warping Distance, DTW).

**Алгоритм DTW**, как следует из названия, пытается деформировать ось времени одного из сигналов(сжатие/растяжение для каждых двух точек). Деформация производится таким образом, чтобы найти минимальное расстояние между двумя сигналами (рисунок приведен ниже). Сам алгоритм, в наиболее общем случае, состоит из следующих шагов: производится расчет матрицы расстояний, таким образом, что для каждой точки одного ряда рассчитываются расстояния до всех точек другого ряда. Обозначим такую матрицу как для двух сегментов: и . производится расчет деформаций временной шкалы по формуле

Среди всех деформаций выбираются пары индексов , для которых выполняется условие . Производится поиск такой траектории деформаций, для которой сумма расстояний была бы минимальным.

Для этого достаточно двигаться по верхнему треугольнику матрицы сверху вниз. На практике мы сможем вычислить сумму как последний элемент , вычисленный по предыдущей формуле.

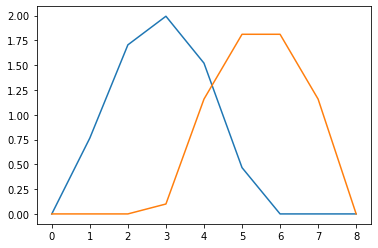
Ниже будет приведен пример реализации данного алгоритма.

import numpy as np  
import matplotlib.pyplot as plt

Зададим два сегмента временного ряда

period = 5  
N = 9  
  
x = np.zeros(N)  
x[1:1+period] = 2\*np.sin(np.pi\*np.arange(period)/period+np.pi/8)  
y = np.zeros\_like(x)  
y[3:3+period] = 1.8\*np.sin(np.pi\*np.arange(period)/period)+0.1  
  
plt.plot(x)  
plt.plot(y)  
plt.show()

Визуализация сегментов приведена ниже.



создадим матрицу расстояний между ними

dist\_matrix = np.zeros((N,N))  
for i,xi in enumerate(x):  
 dist\_matrix[i,:] = (xi-y)\*\*2  
  
print('\n\n Матрица расстояний \n\n')   
print(np.fix(1000\*dist\_matrix)/1000)

Матрица расстояний   
  
  
[[0. 0. 0. 0.01 1.34 3.282 3.282 1.34 0. ]  
 [0.585 0.585 0.585 0.442 0.154 1.095 1.095 0.154 0.585]  
 [2.907 2.907 2.907 2.576 0.299 0.011 0.011 0.299 2.907]  
 [3.975 3.975 3.975 3.586 0.698 0.033 0.033 0.698 3.975]  
 [2.312 2.312 2.312 2.018 0.131 0.084 0.084 0.131 2.312]  
 [0.217 0.217 0.217 0.134 0.477 1.809 1.809 0.477 0.217]  
 [0. 0. 0. 0.01 1.34 3.282 3.282 1.34 0. ]  
 [0. 0. 0. 0.01 1.34 3.282 3.282 1.34 0. ]  
 [0. 0. 0. 0.01 1.34 3.282 3.282 1.34 0. ]]

Теперь создаим процедуру поиска минимальной траектории.

D = np.zeros((N+1,N+1))   
D[0,1:] = np.inf  
D[1:,0] = np.inf  
D[1:,1:] = dist\_matrix  
   
for i in range(1,N+1):  
 for j in range(1,N+1):  
 D[i,j] = D[i,j] + min(D[i-1,j],D[i,j-1],D[i-1,j-1])  
  
   
print('\n\n матрица деформированных расстояний ')  
print(np.fix(1000\*D)/1000)   
print('\n\n')  
print(f'DTW расстояние между сегментами временного ряда {D[N,N]:.3f}')  
print(f'эвклидово расстояние между сегментами временного ряда {np.sqrt(np.sum((x-y)\*\*2)):.3f}')

матрица деформированных расстояний   
[[ 0. inf inf inf inf inf inf inf inf inf]  
 [ inf 0. 0. 0. 0.01 1.35 4.633 7.916 9.257 9.257]  
 [ inf 0.585 0.585 0.585 0.442 0.164 1.259 2.354 2.508 3.094]  
 [ inf 3.493 3.493 3.493 3.019 0.463 0.175 0.186 0.486 3.394]  
 [ inf 7.469 7.469 7.469 6.606 1.162 0.208 0.208 0.885 4.461]  
 [ inf 9.782 9.782 9.782 8.624 1.293 0.293 0.293 0.34 2.653]  
 [ inf 10. 10. 10. 8.759 1.771 2.102 2.102 0.771 0.558]  
 [ inf 10. 10. 10. 8.769 3.112 5.054 5.385 2.112 0.558]  
 [ inf 10. 10. 10. 8.779 4.453 6.395 8.337 3.453 0.558]  
 [ inf 10. 10. 10. 8.789 5.794 7.736 9.678 4.794 0.558]]  
  
  
ИЗ матрицы видим (нижнее правое значение):  
DTW расстояние между сегментами временного ряда 0.558  
эвклидово расстояние между сегментами временного ряда 3.694

Объеденим весь предыдущий опыт в функции

def generate(amplitude = 1, period = 10,   
 phase\_shift = 0,   
 bias = 0, N = 100,   
 start= 0, end = None):  
 if not end: end = N  
 x = np.zeros(N)  
 x[start:start+end] = bias + amplitude\*np.sin(2\*np.pi\*np.arange(end)/period+phase\_shift)  
 return x  
  
def euclidian\_matrix(x,y):  
 dist\_matrix = np.zeros((x.size,y.size))  
 for i,xi in enumerate(x):  
 dist\_matrix[i,:] = (xi-y)\*\*2  
 return dist\_matrix  
   
def dtw(x, y):  
  
 N = x.shape[0]  
  
 n\_rows = n\_columns = N + 1  
   
 D = np.zeros((n\_rows,n\_columns))  
  
 D[0,1:] = np.inf  
 D[1:,0] = np.inf  
  
 D[1:,1:] = euclidian\_matrix(x,y)   
   
 for i in range(1,n\_rows):  
 for j in range(max(1, i-N),n\_columns):  
 D[i,j] = D[i,j] + min(D[i-1,j],D[i,j-1],D[i-1,j-1])  
  
 return D[N,N]  
  
def euclidian\_distance(x,y):  
 return np.sqrt(np.sum((x-y)\*\*2))

dtw(x,y)

0.558248747803976

Сверим результат на соответствие с функцией из стандартной библиотеки [sktime](https://github.com/alan-turing-institute/sktime/blob/84852f64e78961f1a4b2372731241457b1106587/sktime/distances/_dtw.py).

from sktime import distances

distances.dtw\_distance(x,y)

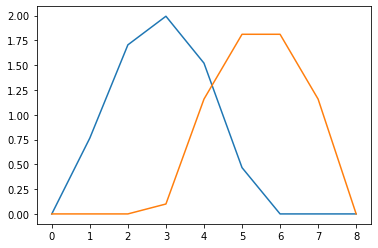
0.558248747803976

print(f'DTW расстояние между сегментами временного ряда {dtw(x,y):.3f}')  
print(f'эвклидово расстояние между сегментами временного ряда {euclidian\_distance(x,y):.3f}')

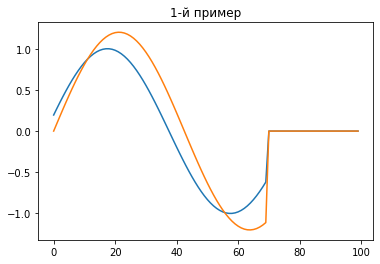
DTW расстояние между сегментами временного ряда 0.558  
эвклидово расстояние между сегментами временного ряда 3.694

Посмотрим также на примера сгенерированных сегментов, таких, что во втором премере сегмент

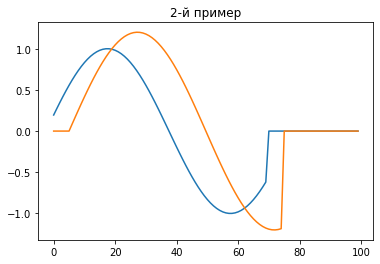
x = generate(amplitude = 1, period = 80,   
 phase\_shift = np.pi/16,   
 bias = 0, N = 100,   
 start = 0, end = 70)  
  
y = generate(amplitude = 1.2, period = 85,   
 phase\_shift = 0,   
 bias = 0, N = 100,   
 start = 0, end = 70)  
plt.plot(x)  
plt.plot(y)  
plt.title('1-й пример')  
plt.show()  
  
print(f'DTW расстояние между сегментами временного ряда {dtw(x,y):.3f}')  
print(f'эвклидово расстояние между сегментами временного ряда {euclidian\_distance(x,y):.3f}')  
  
x = generate(amplitude = 1, period = 80,   
 phase\_shift = np.pi/16,   
 bias = 0, N = 100,   
 start = 0, end = 70)  
  
y = generate(amplitude = 1.2, period = 89,   
 phase\_shift = 0,   
 bias = 0, N = 100,   
 start = 5, end = 70)  
  
plt.plot(x)  
plt.plot(y)  
plt.title('2-й пример')  
plt.show()  
  
print(f'DTW расстояние между сегментами временного ряда {dtw(x,y):.3f}')  
print(f'эвклидово расстояние между сегментами временного ряда {euclidian\_distance(x,y):.3f}')



DTW расстояние 0.558   
эвклидово расстояние 3.694



DTW расстояние между сегментами временного ряда 1.624  
эвклидово расстояние между сегментами временного ряда 2.379



DTW расстояние между сегментами временного ряда 1.940  
эвклидово расстояние между сегментами временного ряда 5.481

Данный пример показателен в отношении стабильности значений расстояний DTW к деформациям сигнала.

Особенности расстояния DTW – это стабильность значений к деформациям ВР, смещениям и т.д., но расстояние не линейное, То есть при смещении одного сегмента относительно другого значение расстояния может изменяться не пропорционально .

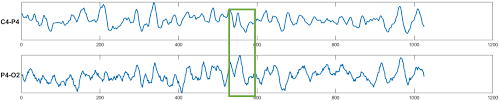
Отметим, что помимо классического DTW существуют его модифицированные версии, а также и другие варианты эластичных мер. В некоторых случаях для классификации используются ансамбли различных эластичных мер.

Также важно заметить, что ряд исследователей предлают использовать алгоритм 1-ближайший сосед с DTW расстоянием (1NN-DTW) в качестве базового результата (baseline) для решения задач классификации временных рядов.

# Юнит 21 - Другие методы на основе сегментов данных

Ряд исследователей относят оценки расстояний по эластичным мерам т.н. "whole series". В противоположность к таким подходам авторы выделяют "intervals" оценки - то есть оценки по одному или несколькими частям сегмента временного ряда. При этом сегменты могут быть выбраны:

* в соответствии с каким-либо критерием (например, как первое и последнее значения, превышающие заданное пороговое значение);
* как некоторый, случайным образом выбранный интервал,
* Путем выбора лучшего из случайных интервалов;
* Могут быть выбраны несколько интервалов для каждого сегмента.



Пример выбора сегмента в данных разных классов как некоторого «окна» в заданный момент времени.

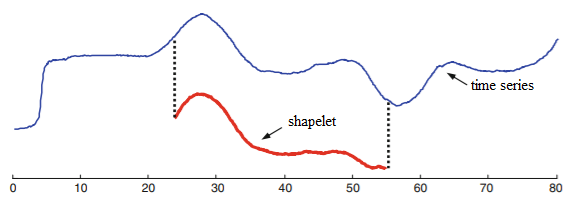
Среди достоинств интервальных оценок по сравнению с оценками на основе эластичных мер можно выделить следующие:

* Ускоренное время работы алгоритма. В отличии от, например, 1NN-DTW тут в работа осуществляется только с небольшим участком временного ряда.
* Более высокая интерпретируемость результатов. В отличии от, например, 1NN-DTW в данном подходе дискриминативная способность каждого участка ряда может быть оценена визуально или напрямую сравнена с другим участками. Однако процедура поиска лучшего участка ряда может стать отдельной и достаточно вычислительно-сложной задачей.



Пример выбора интервала, по степени различия для класса 0 (сверху) и класса 1 (средний и нижний графики).

Одной из наиболее популярных реализаций, описанных выше идей является использование т.н. "Шейплетов"(Shapelet). Шейплет является сравнительно коротким участком (паттерном) временного ряда, который в наибольшей степени позволяет отличить один класс от другого. Такая близость может быть оценена метрически (эвклидово расстояние) или по любой другой заданной мере схожести.



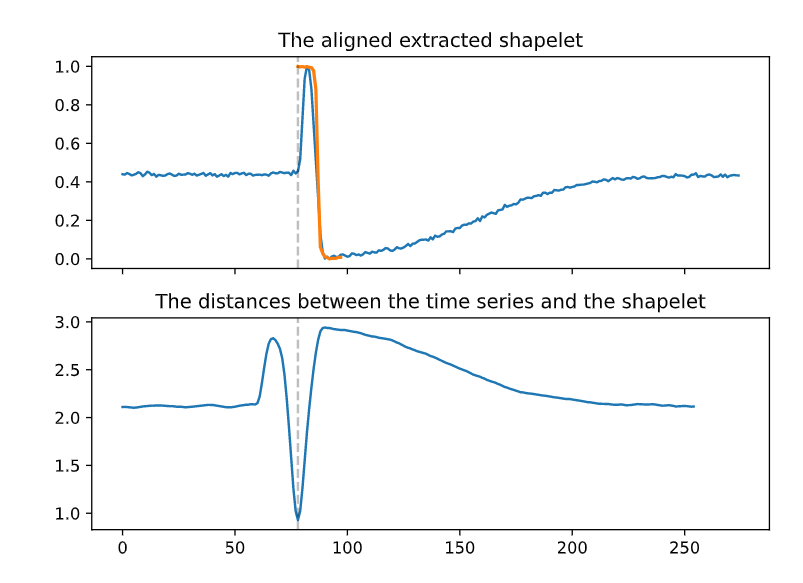
Пример шейплета – участок, который, как мы ожидаем в наибольшей степени характеризует сегмент ВР.

Шейплеты могут быть расположены в различных участках временных рядов. В некоторых случаях может быть осуществлен поиск более одного шейплета для решения задачи.

Шейплеты могут быть найдены, например, перебором по сетке значений (brute force), поиском по максимуму расстояния между классами (минимуму расстояний внутри класса) или при других методов оптимизации. Для найденных шейплетов классификация может быть проведена аналогично как по сегментам (например, методов 1NN-DTW).

Отметим, что процедура поиска и выделения шейплетов может также быть рассмотрена отдельно от классификации, в этом случае ее еще называют "Shapelet Transform".

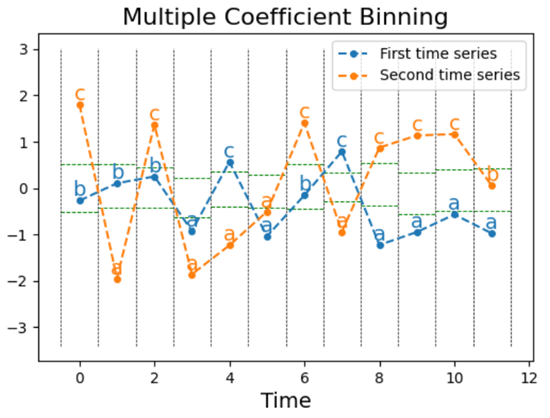
[Пример шейплетов для двух классов](https://tslearn.readthedocs.io/en/stable/user_guide/shapelets.html). Выделенный участок в наибольшей степени позволяет различить классы



Относительными недостатками подхода шейплетов является неспособность работать в тех случаях, когда важно не само наличие какого-либо характерного участка в сегменте ряда (паттерна), а, например, частота появления участка.

Если наиболее важным является вопрос о поиске не самого паттерна во временном ряду, а, например, частота и характер появления некоторых его особенностей. В таких и подобных задачах для описания временных рядов (для их представления) могут быть использованы подходы на основе т.н. словаря (Dictionary-based).

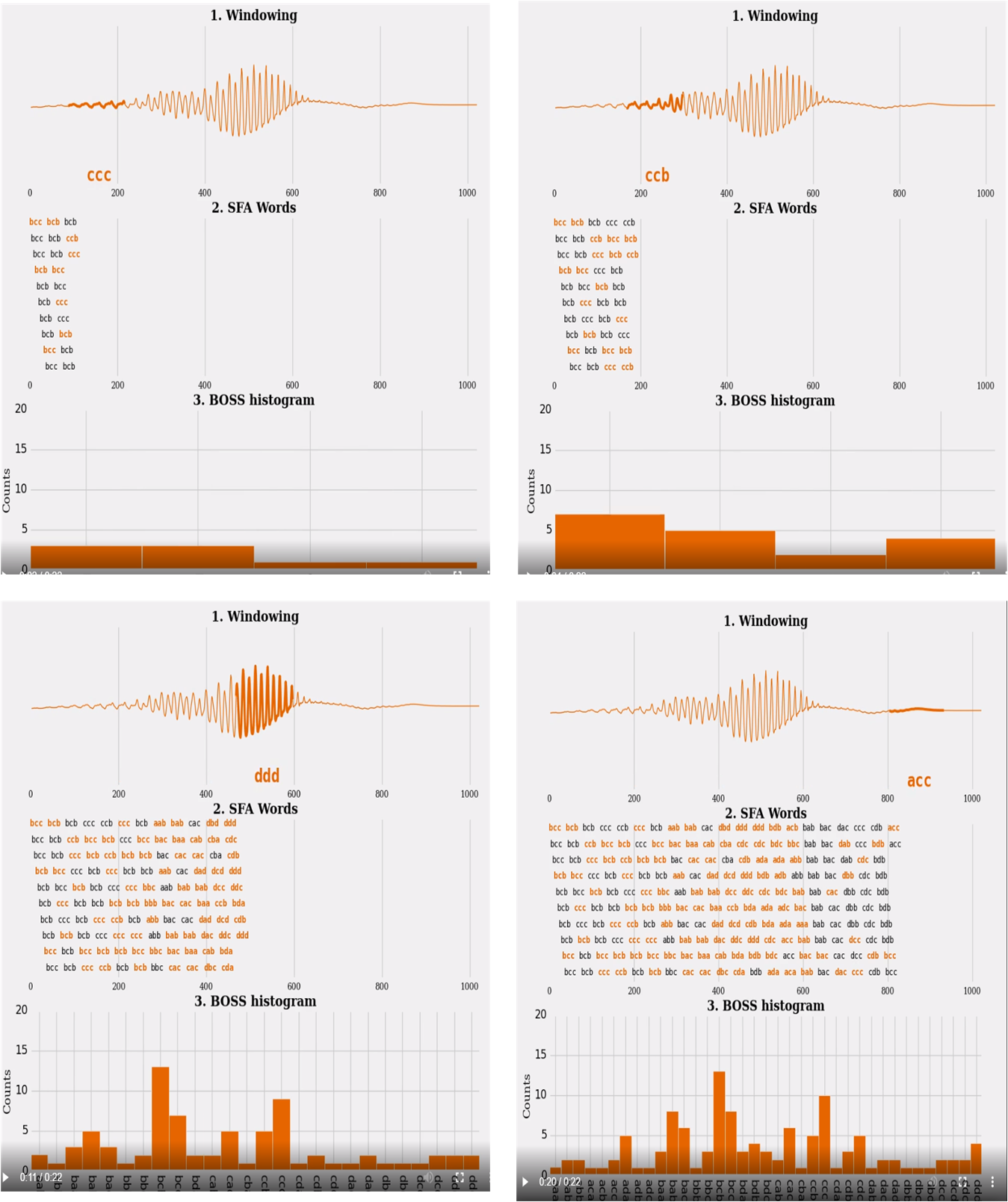
Суть алгоритмов на основе словаря сводится к следующему. По сегменту временному ряда скользит окно заданной длины (происходит выборка участков). Для каждого участка осуществляется "грубая" аппроксимация, как по шкале времени (дискретизация), так и по шкале значений ряда (квантование). Например, [на рисуке](https://johannfaouzi.github.io/pyts/auto_examples/plot_sax.html) ниже выделено 3 уровня квантования (a, b и c) и 12 шагов по времени. Хотя реальный ВР мог иметь и меньший шаг и значения по амплитуде.



Для каждого участка формируется т.н. "слово". Например, для синего класса на рисунке слово будет "bbbacabcaaaa". Слово всегда формируется из т.н. "алфавита" заданного заранее. То есть каждому из трех квантов амплитуды мы поставили свой символ «a», «b» и «c». Затем мы составили из символов для каждого участка слово. Мы могли бы разделить слово на несколько, например, по 3 шага, тогда у нас было бы 4 сегмента вида "bbb aca bca aaa". Слова могут выделяться для разных трансформаций участка. Например, по результату разложения участка на тренд и сезонность. Таким образом, можно для каждого сегмента набрать целый «мешок» слов. Для полученных слов рассчитывается гистограмма их появлений для каждого временного ряда. В какой то степени это некоторое отображение ВР.

Для таких отображений – гистограмм можно рассчитать различия. Мы назовём их расстояниями. Расстояние между гистограммами отдельных временных рядов (их пересечение) может являться признаком для их классификации.

Пусть мы движемся окном по сегменту ВР. Для каждого положения окна описание ряда может быть дано в нескольких частотных полосах – то есть мы провели разложение на компоненты ряда Фурье. Для каждой компоненты построим сове словарное описание. На этом принципе построена одна из наиболее популярных техник на основе словарей - BOSS (Bag of SFA Symbols). Где SFA (Symbolic Fourier Approximation) это тип аппроксимации по ряду Фурье (на самом деле в частотной области). [Пример построения гистограмм при помощи BOSS](https://www.researchgate.net/publication/327762252_Human_Activity_Recognition_on_Smartphones_using_Symbolic_Data_Representation/figures?lo=1) приведен на рисунке ниже справа.



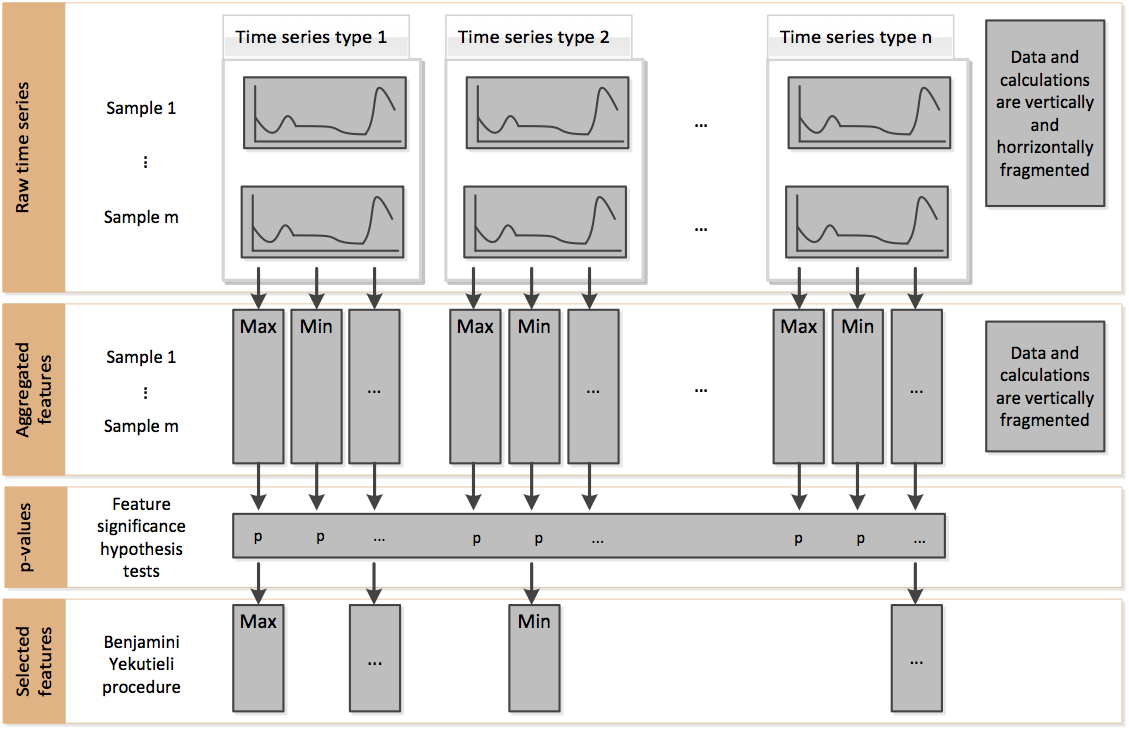
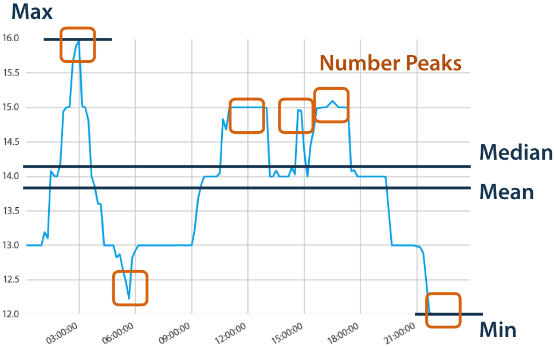
В практическом юните мы увидим, как это работает.

# Юнит 22 - Методы на основе признаков.

Подход на основе признаков предполагает, что модель строится на выделение определенных регулярных характеристиках для каждой последовательности.

Признаки:

* Регулярны для набора данных.
* Отражают класс (регулярны для класса).
  + Экзогенные факторы – тоже могут быть признаками.
* Не коррелируют друг с другом (иначе избыточны!).
  + Можно решить отбором признаков.
* Позволяют различать классы как можно более четко.
  + А еще можно преобразовать признаки.
* Пространство признаков должно быть достаточным для классификации.
  + От этого будет зависеть точность классификации



Иллюстрации признаков ВР.

Признаки могут быть выделены для всего сегмента ряда (глобальные признаки) или для некоторых его частей (интервальные признаки). При этом интервалы могут быть как детерминированными, так и выбранными случайно, причем разной длины.

Признаки могут быть: точечными (одно значение), векторными (например набор коэффициентов авторегрессии).

Признаки могут быть выделены: как некоторые суммарные статистики; во временной области или в частотной области. А также по результатам некоторого время-частотного представления сегмента ряда.

Отметим также, что перед выделением признаков ряд может быть предобработан. Признаки многомерного ВР могут быть выделены в едином пространстве или для каждой компоненты ВР свои, также результаты для всех компонент могут быть объединены. После выделения признаки могут быть отобраны и трансформированы.

Если временной ряд многомерный, то есть каждый сегмент временного ряда содержит несколько одномерных составляющих. В таких случаях могут быть несколько подходов к выделению признаков: объединение признаков нескольких составляющих в один вектор; ассемблирование результатов классификации по каждой составляющей; использование специальных методов выделения признаков многомерных рядов, например, при помощи векторной авторегрессии.

Результат работы с признаками – это всегда некоторая таблица. На ее основе может быть построен классификатор. Могут быть использованы как стандартные классификаторы, например, XGBoost, так и специализированные.

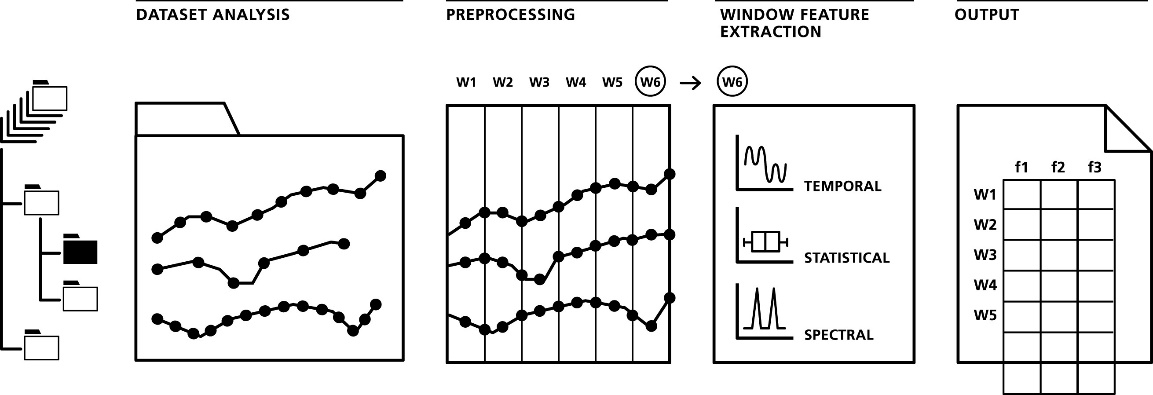


Иллюстрация пайплайна работы с многомерными признаками.

Часто достаточно выделить лишь наиболее простые статистические признаки, такие как среднее или стандартное отклонение для каждого сегмента временного ряда. Однако, потенциально, число допустимых признаков для временного ряда может быть достаточно большим. Поэтому исследователю следует использовать или готовые схемы или известные методы отбора признаков.

Самый простой способ выбора признаков - расчет среднего и дисперсии для каждого сегмента ряда, и использование традиционных классификаторов, например, Логистической регрессии, для полученной таблицы (каждая запись: признаки, метка класса).

Вот примеры некоторых признаковых пространств.

* + **Описательные статистики**
    - среднее, СКО, дисперсия,
    - Моменты высших порядков,
    - Другие параметры распределений: медиана, мода, перцентиль, энтропия и т.д..
    - Значения статистических тестов, напр. ADF как хар-р стационарности.
    - Результат сравнения с эталоном по метрике (например коэффициент корреляции).
  + **Временные параметры**
    - макс, мин, среднее значения,
    - Наклон тренда и его параметры,
    - Параметры декомпозиции (сезон, число сезонностей),
    - Расстояние между особыми точками (пиками, нулями).
    - Значения лагов АКФ, ЧАКФ
    - Параметры ошибки разложения или предсказания заданной моделью.
* **Спектральные параметры** 
  + Макс. Частота (период ВР), интенсивности отдельных составляющих ряда Фурье,
  + Площадь под кривой
  + Спектральные статистики
  + Значения главных компонент (метод PCA),
  + Коэффициенты AR или ARIMA коэффициенты.
* **Время-частотные, псевдо спектральные**
  + *Аналогичные спектральным признаки, но полученные для каждого положения выбранного окна, которое движется по ВР (*время-частотногоразложения).
  + вейвлетразложения и их типы и другие

Достоинствами подхода на основе признаков являются потенциальная возможность создание классификатора с минимальной избыточностью относительно решаемой проблемы и при этом с высокой обобщающей способностью. Однако, следует понимать, что в ряде случаев формализация признаков и поиск подходящего признакового пространства может представлять отдельную, достаточно сложную задачу.

На практике при работе с признаками рекомендуется начинать с наиболее интерпретируемых и понятных точечных признаков. То есть двигаться от простого к сложному.

Во многих случаях для поиска лучшего признакового пространства существуют готовые фреймворки. Примерами таких для языка Python могут быть:

* [tsfresh](https://github.com/blue-yonder/tsfresh)(<https://tsfresh.readthedocs.io/en/latest/>);
* [tsfel](https://github.com/fraunhoferportugal/tsfel)(<https://tsfel.readthedocs.io/en/latest/>);
* [Catch22](https://github.com/DynamicsAndNeuralSystems/catch22)(<https://github.com/DynamicsAndNeuralSystems/catch22/wiki>);
* и многие другие.

В большинстве случаев такие Фреймворки предоставляют как возможности для выделения признаков, так и для их отбора.

Несмотря на замечание о возможности использования классификаторов общего назначения укажем несколько примеров специальных алгоритмов для работы с временными рядами. Примеры реализации данных алгоритмов могут также быть найдены в sktime.

Одним из наиболее популярных методов анализа временных рядов на основе признаков является т.н. Time Series Forest (TSF). В основе данного подхода лежит следующая последовательность действий.

* Для каждого сегмента временного ряда выделяется набор интервалов, выбранных случайно.
* Для каждого интервала производится оценка таких признаков, как среднее значение, стандартное отклонение и наклон линейного тренда. Возможны и другие характеристики.
* Для признаков каждого интервала строится отдельное дерево.
* Среднее значение (либо голосование) по ансамблю деревьев рассматривается как результат работы алгоритма.

В оригинальной работе авторы также предложили особый критерий расщеплений в дереве. Отметим, что в общем случае алгоритм TSF не ограничивается предложенными авторами признаками. Например, авторы подхода Catch22 предложили название использования Canonical Interval Forest (CIF) для использования их набора признаков совместно с подходом TSF для построения деревьев.

Алгоритмы на основе TSF показывают высокие точности для ряда задач. Однако, метод описывает лишь временные характеристики ряда не позволяя учитывать спектральные составляющие.

Другим примером использования деревьев во временных рядах является подход RISE - Random Interval Spectral Ensemble данный подход был предложен как учитывающий, как глобальные признаки (по всему сегменту временного ряда), так и локальные (полученные по случайным интервалам в рамках сегмента). Для каждого участка ряда выделяются такие векторные признаки, как: набор нескольких значений автокорреляционной функции; набор нескольких значений спектральной мощности (значений амплитудного спектра); набор нескольких коэффициентов авторегрессионной функции.

Отметим, что последний набор признаков может быть опущен. Так это сделано, в реализации пакета [sktime](https://github.com/alan-turing-institute/sktime/blob/7be01f62e580db77da1420823291d5d03675b45d/sktime/classification/interval_based/_rise.py). Для каждого случая выбранного пространства признаков строится отдельное дерево. В оригинальной статье авторы рекомендуют строить одно глобальное дерево (по всему сегменту ряда) и локальных деревьев.

~~В конце раздела отметим некоторые полезные материалы по теме:~~ [~~https://www.timeseriesclassification.com/~~](https://www.timeseriesclassification.com/) ~~и особенно раздел~~ [~~алгоритмы~~](https://www.timeseriesclassification.com/algorithm.php)~~, который дает некоторое представление о существующем прогрессе в этой области на сегодня. раздел веб-сайта~~ [~~https://paperswithcode.com/task/time-series-classification~~](https://paperswithcode.com/task/time-series-classification) ~~, который дает некоторое представление о существующих бенчмарках, и наиболее точных подхах для них. Однако, тут приемущественно можно найти Deep Learning-based подходы. раздел фреймворка SKTime~~ [~~https://www.sktime.org/en/stable/api\_reference/classification.html~~](https://www.sktime.org/en/stable/api_reference/classification.html)~~, содержащий реализации популярных классификаторов временных рядов.  
 библиотека TSAI~~ [~~https://timeseriesai.github.io/tsai/~~](https://timeseriesai.github.io/tsai/)~~, которая включает реализацию ряда глубоких нейронных сетей для работы с временными рядами в рамках фреймврока PyTorch.  
 Также бы хотелось отметить, что часто сравнение классических алгоритмов производится для архива наборов данных~~ [~~UCR Time Series Classification Archive~~](https://www.cs.ucr.edu/~eamonn/time_series_data/)~~, последняя версия самого архива может быть получена по~~ [~~ссылке~~](https://www.cs.ucr.edu/~eamonn/time_series_data_2018/)~~. Архив содержит более 100 наборов.~~

# Юнит 23 - Комбинации подходов

Напомним, что если признаками не удается полностью формализовать решение задачи, то, полученное признаковое пространство может привести к потере точности и/или обобщающей способности.

Для таких задач, на практике, методы на основе данных и на основе признаков могут быть объединены в ансамбли. Цель - учесть преимущества и недостатки тех и других подходов.

При этом надо отметить, что часто в анализе временных рядов ансамблевые методы предполагают использование различных подходов, решающих разные типы задач.

Например, в ансамбль объедены могут быть подходы:

* ориентированные на работу с временными признаками (такие, как TSForest);
* ориентированные на работу в спектральной области (такие, как RISE);
* ориентированные на обработку некоторых шаблонов формы (такие, как Шейплеты);
* и ориентированные на обработку повторяющихся шаблонов формы (такие, как словари BOSS).

Примером реализации такого ансамбля является алгоритм HIVE-COTE (The Hierarchical Vote Collective of Transformation-based Ensembles with Collective of Transformation-based Ensembles). Ансамбль представляет собой набор результатов, полученных для нескольких вариантов каждого из указанных подходов и объединённых с весовыми параметрами, определяемыми в ходе обучения.

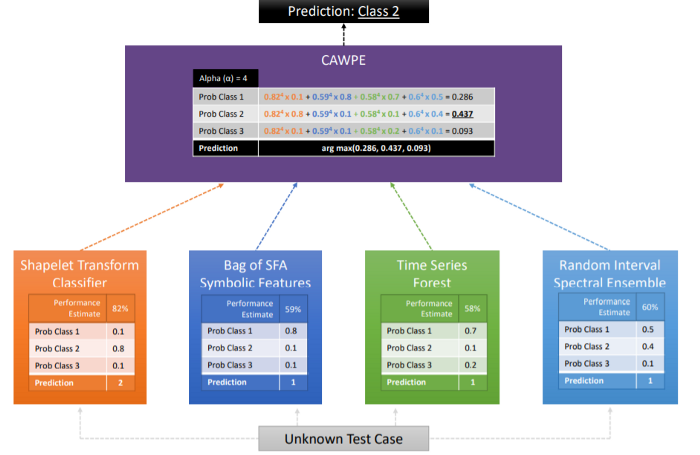


Иллюстрация алгоритма Hive-COTE

Прогнозы HIVE-COTE представляют собой средневзвешенное значение прогнозов, сделанных его участниками: шейплеты, словари, лес временных рядов, метод RISE. Каждый подклассификатор оценивает вероятность каждого класса отдельно. Блок управления объединяет эти вероятности с обучаемыми всеми (CAPWE). Веса назначаются как относительная оценка качества классификатора,   
найденного на обучающих данных.

Метод HIVE-COTE – это один из наиболее точных и ярких представителей идеи ассемблирования в классификации временных рядов. Метод работает на широком круге задач, где как правило показывает лучшие результаты. Даже лучше отдельных глубоких нейронных сетей. Однако, метод требует значительных вычислительных ресурсов для обучения и времени работы. Метод тут приведен именно как иллюстрация того принципа что во временных рядах часто можно выделить признаки вручную и можно построить классификатор, вручную не прибегая к глубокому обучению нейронных сетей. Однако это может привести к росту затрат в сравнении с теми же нейронными сетями.

Другой метод классификации который мы хотели бы тут показать напротив имеет сравнительно низкую вычислительную сложность, обладая сопоставимой мощностью классификации. Это метод на основе случайных сверток - т.н. ROCKET классификатор (RandOm Convolutional KErnel Transformation classifier).

Идея подхода ROCKET появилась сравнительно не давно и основана на сверточных нейронных сетях. Сама по себе процедура классификации состоит из следующих шагов. Производится выбор параметров случайных параметров расширенного одномерного сверточного слоя нейронной сети. В том числе:

* длина ядра (число значений),
* значения весовых параметров,
* смещение ядра,
* степень расширения свертки (dilation rate)
* и добавление нулей (padding).

Как правило число генерируемы таким образом ядер сверток около . Для результата воздействия на сегмент каждого фильтра выбираются максимальное значение и отношение числа значений больше нуля к длине сегмента (ppv - proportion of positive values).

В качестве итогового алгоритма авторы оригинальной статьи предлагали использовать или Гребневую регрессию с кросс-валидацией (RidgeRegressionCV) или логистическую регрессию (LogisticRegression) для достаточно больших наборов данных.  
  
Во многих задачах подход ROCKET является достаточно точным и при этом показывает высокую скорость как в работе, так и в обучении. Однако подход практически не воспроизводимый. То есть два раза получить тот же набор ядер не удастся. Поэтому тут есть проблемы с повторным обучением и прочими связанными проблемами.

Заметим, что одна из модификаций алгоритма (от авторов оригинальной работы) - MiniRocket сокращает число возможны вариантов генерации сверточных ядер, что ускоряет обучение в порядка раз без значительных потерей точности. Это делает подход более вопроизводимым.

Один из вариантов подходов на основе ROCKET (Arsenal) был также включен во вторую версию ансамблей HIVE-COTE2 наряду с особой версией CIF, алгоритмами на базе словорей и алгоритмами на основе шейплетов.

# Юнит 24 – Классификация в SKTime.

Классификация в SKTime

Импорт библиотек

Начнем с импорта нужных библиотек

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt

try:  
 import sktime  
except:  
 !pip install sktime --user  
 !pip install tsfresh   
 !pip install numba  
 !pip install llvmlite  
import sktime

from sktime.transformations.panel.rocket import MiniRocket

from sktime.datasets import load\_from\_tsfile  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.model\_selection import train\_test\_split

## Импорт данных

Теперь выберем набор данных. Для нашего урока мы воспользуемся уже имеющимся набором данных [открытые данные энергетических систем](https://open-power-system-data.org/).

url = 'https://data.open-power-system-data.org/time\_series/2020-10-06/'  
datafile = url + 'time\_series\_60min\_singleindex.csv'  
df\_all = pd.read\_csv(datafile, index\_col='utc\_timestamp', parse\_dates=True, low\_memory=False)

Также как и ранее выберем нужную страну, для примеров это была Германия.

def extract\_country(df\_all, country\_code, year\_min=None, year\_max=None):  
 """Extract data for a single country"""  
   
 # List of columns to extract  
 columns = [col for col in df\_all.columns if col.startswith(country\_code)]  
   
 # Extract columns and remove country codes from column labels  
 columns\_map = {col : col[3:] for col in columns}  
 df\_out = df\_all[columns].rename(columns=columns\_map)  
   
 # Exclude years outside of specified range, if any  
 if year\_min is not None:  
 df\_out = df\_out[df\_out.index.year >= year\_min]  
 if year\_max is not None:  
 df\_out = df\_out[df\_out.index.year <= year\_max]  
   
 return df\_out  
  
def transform\_dataframe(df, cols\_map):  
 # Rename columns for convenience  
 df = df[list(cols\_map.keys())].rename(columns=cols\_map)  
 # Convert from MW to GW  
 df = df / 1000  
# df = df.groupby(df.index.hour).mean()  
 df = df.rename\_axis('Date')  
# df.index = df.index.strftime('%Y-%m-%d-%h')  
 return df

df\_hrly = extract\_country(df\_all, country\_code='DE', year\_min=2015, year\_max=2019)  
df\_hrly.sample(3)  
  
cols\_map = {'load\_actual\_entsoe\_transparency' : 'Consumption',  
 'wind\_generation\_actual' : 'Wind',  
 'solar\_generation\_actual' : 'Solar'}  
df = transform\_dataframe(df\_hrly, cols\_map).dropna()  
  
# df wind + solar generation  
df['Wind+Solar'] = df[['Wind', 'Solar']].sum(axis=1, skipna=False)  
df.to\_csv('de\_clf\_data.csv')  
df.head()

Consumption Wind Solar Wind+Solar  
Date   
2015-01-01 07:00:00+00:00 41.133 10.208 0.071 10.279  
2015-01-01 08:00:00+00:00 42.963 10.029 0.773 10.802  
2015-01-01 09:00:00+00:00 45.088 10.550 2.117 12.667  
2015-01-01 10:00:00+00:00 47.013 11.390 3.364 14.754  
2015-01-01 11:00:00+00:00 48.159 12.103 4.198 16.301

Однако для текущей цели - изучить классификацию временных рядов давайте сделаем набор данных с классами. Для этого предлагается поставить задачу определение по дневному потреблению сезона (зима, весна, осень, лето):

* Разделить все данные на 4 сезона (зима, весна, осень, лето)
* взять данные с шагом 1 час.
* создать сегменты размером 24 часа (1 день).

Данная задача полностью искусственная, однако в целом соответствует некоторым из задачи стандартных тестов классификации временных рядов, например смотрите [тут](http://www.timeseriesclassification.com/description.php?Dataset=PowerCons).

Напомним, что для проведения классификации, как правило, данные следует разделять на сегменты. Как правило, эти сегменты не должны пересекаться. В нашем случае длина сегмента будет 24 точки.

Для цели создания набора данных давайте создадим 2 функции, первая split\_by\_month будет разделять наш набор данных на месяца. Вторая функция to\_segments будет создавать сегменты по 24 часа в сутках.

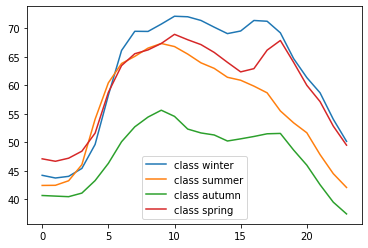
def split\_by\_month(df, months):  
 df\_ = pd.DataFrame()  
 for month in np.atleast\_1d(months):  
 df\_ = df\_.append(df.loc[df.index.month == month])  
 return df\_  
  
def to\_segments(df, column, size = 24):   
 df.index.hour[0]  
 start\_idx = 24-df.index.hour[0]  
 df = df.iloc[start\_idx:]  
 val = df[[column]].values  
 return val[:size\*(val.size//size)].reshape(-1,size)

winter = to\_segments(split\_by\_month(df, [1,2, 12]), 'Consumption', size = 24)  
summer = to\_segments(split\_by\_month(df, [6, 7, 8]), 'Consumption', size = 24)  
autumn = to\_segments(split\_by\_month(df, [9,10,11]), 'Consumption', size = 24)  
spring = to\_segments(split\_by\_month(df, [3, 4, 5]), 'Consumption', size = 24)  
print(winter.shape, summer.shape, autumn.shape, spring.shape)

(448, 24) (458, 24) (452, 24) (458, 24)

Теперь мы создали 4 класса. Давайте посмотрим как выглядят их примеры.

plt.figure()  
day = 10  
for i,(c,d) in enumerate(zip([winter,summer,autumn,spring],['winter','summer','autumn','spring'])):  
 plt.plot(c[day], label="class " + str(d))  
plt.legend(loc="best")  
plt.show()  
plt.close()



Хотя примеры показывают высокую степень сходства мы все же попробуем их классифицировать. Для этого создадим массив тренировочных и тестовых данных.

X = np.concatenate((  
 winter,   
 summer,   
 autumn,   
 spring))  
y = np.concatenate((  
 0\*np.ones(winter.shape[0]),  
 1\*np.ones(summer.shape[0]),  
 2\*np.ones(autumn.shape[0]),  
 3\*np.ones(spring.shape[0])  
 ))  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
print(X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape)

(1271, 24) (1271,) (545, 24) (545,)

Современная версия SKTime допускает формат массивов в виде 2d ndarray (numpy) для однопеременных временных рядов. Для многопеременных временных рядов массив может быть задан в виде 3d ndarray. В общем случае однопеменный ряд также может быть задан как многопеременный с размерностью массива Также в пакете SKTime предусмотрены и другие форматы. В том числе специализированные. Для преобразования между форматами используются convert и convert\_to из модуля sktime.datatypes. Например, для преобразования двухмерного массива во встроенный однопеременных формат можно воспользоваться следующим скриптом

df = convert\_to(np.atleast\_3d(X).transpose(0,2,1), to\_type="nested\_univ")

Однако не все функции SKTime поддерживают 2d ndarray поэтому давайте создадим nested\_univ копии данных.

from sktime.datatypes import convert\_to  
df\_train = convert\_to(np.atleast\_3d(X\_train).transpose(0,2,1), to\_type="nested\_univ")   
df\_test = convert\_to(np.atleast\_3d(X\_test ).transpose(0,2,1), to\_type="nested\_univ")

## Классификация sklearn

Перед тестированием специализированных методов из пакета SKTime давайте посмотрим на результаты стандартных подходов из sklearn. В анализе временных рядов среди таких стандартных подходов наиболее распространен случайный лес. Также давайте попробуем алгоритм k-соседей с евклидовым расстоянием как некоторую предварительную базовую оценку.

from sklearn.ensemble import RandomForestClassifier  
from sklearn.neighbors import KNeighborsClassifier  
  
clf = RandomForestClassifier(n\_estimators=100)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.881

clf = KNeighborsClassifier(n\_neighbors=1)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.879

В теоретической части мы уже обсуждали типы алгоритмов классификации временных рядов. Теперь давайте попробуем их на практике. Ранее мы обсуждали что в качестве некоторой базовой оценки можно использовать 1-NN DTW классификатор. Напомним, что этот подход сравнивает временные ряды "в целом". Расстояние DTW относится к эластичным мерам, стабильным к искажениям различных сегментов ряда.

В пакете SKTime данный тип алгоритмов находится в модуле distance\_based.

from sktime.classification.distance\_based import KNeighborsTimeSeriesClassifier  
clf = KNeighborsTimeSeriesClassifier(n\_neighbors=1, distance="dtw")  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.839

Полученный результат не превзошёл результат с евклидовым расстоянием. Вероятно, это связано с близостью форм сегментов. Однако 1-NN DTW может быть реализован и в более продвинутых формах. Одной из таких форм является расстояние разностей ddtw.

from sktime.classification.distance\_based import KNeighborsTimeSeriesClassifier  
  
clf = KNeighborsTimeSeriesClassifier(n\_neighbors=1, distance="ddtw")  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.917

Другим вариантом классификатора на основе данных являются шейплет классификатора. Напомним, что шейплет представляет собой участок временного ряда, обладающий наибольшей дискриминационной способностью для задачи. Другими словами, для участков шейплетов различия между классами наиболее существенны.

В ракете SKTime шейплеты могут быть найдены в модуле sktime.classification.shapelet\_based.

from sktime.classification.shapelet\_based import ShapeletTransformClassifier  
  
clf = ShapeletTransformClassifier(estimator=RandomForestClassifier(n\_estimators=100),  
 n\_shapelet\_samples=100,  
 max\_shapelets=100,  
 batch\_size=20)  
  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.916

Классификаторы на основе шейплетов позволяют найти наиболее различающиеся паттерны в данных. Как обсуждалось в теоретической части в ряде случаев более полезным оказываться повторяемость такие паттернов. В таких и подобных задачах для описания временных рядов (для их представления) могут быть использованы подходы на основе т.н. словаря (Dictionary-based).

В пакете SKTime для работы со словарями используется модуль sktime.classification.dictionary\_based.

from sktime.classification.dictionary\_based import (ContractableBOSS,   
 IndividualBOSS,   
 IndividualTDE,   
 MUSE, WEASEL)

clf = IndividualBOSS(random\_state=47)  
clf.fit(X\_train, y\_train)  
clf.score(X\_test, y\_test)

0.8110091743119267

clf = IndividualTDE(random\_state=47)  
clf.fit(X\_train, y\_train)  
clf.score(X\_test, y\_test)

0.8403669724770643

clf = MUSE(random\_state=47)  
clf.fit(X\_train, y\_train)  
clf.score(X\_test, y\_test)

0.9504587155963303

## Классификаторы на основе признаков

Классификаторы, рассмотрены выше можно было отнести к т.н. классификаторам на основе данных. Однако как уже обсуждалось в теоретической части, часто следует отдавать предпочтения классификаторам на основе признаков. Одними из наиболее популярных подходов среди таких классификаторов являются интервальные подходы. Суть таких подходов сводится к

* поиску или случайному выбору интервалов внутри временных рядов;
* извлечению некоторого набора признаков для такого интервала;
* построению некоторого классификатора, например, дерева для каждого интервала;
* объединение результатов для всех интервалов, например, при помощи леса деревьев.

В пакете SKTime такие методы могут быть найдены в модуле sktime.classification.interval\_based.

from sktime.classification.interval\_based import (TimeSeriesForestClassifier,   
 RandomIntervalSpectralEnsemble,   
 SupervisedTimeSeriesForest)

clf = TimeSeriesForestClassifier(n\_estimators=100,random\_state=47)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.947

clf = RandomIntervalSpectralEnsemble(n\_estimators=100, random\_state=47)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.934

clf = SupervisedTimeSeriesForest(n\_estimators=100, random\_state=47)  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.947

Помимо алгоритмов на основе деревьев, популярным на сегодня является методы на основе ядерных преобразований. Одним из наиболее успешных алгоритмов в этом направлении является алгоритм Rocket. Суть данного алгоритма сводится к трансформации данных при помощи набора сверточных ядре со случайно выбранными параметрами. Затем авторы рекомендуются использовать RidgeClassifierCV, который предоставляет собой Логистическую регрессию с L2 регуляризацией, параметр регуляризации выбирается при помощи кросс-валидации.

Мы рассмотрим ускоренную версию преобразования MiniRocket.

from sklearn.linear\_model import RidgeClassifierCV  
  
transform = MiniRocket(num\_kernels=10000, n\_jobs = -1)  
Xt\_train = transform.fit\_transform(X\_train)  
Xt\_test = transform.transform(X\_test)  
  
clf = RidgeClassifierCV()  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(X\_test, y\_test):.3f}')

test score 0.912

Другим важным подходом на основе признаков является подход выбора признакового пространства. Таким пространством может быть лишь использование небольшого числа суммарных статистик (среднее, медиана, дисперсия и т.д.), которые должны как-то описать временной ряд.

В некоторых случаях суммарных статистик может быть недостаточно. В общем случае помимо суммарных статистик могут быть и Временные или частотные (псевдочастотные) признаки. Давайте перечислим примеры таких признаков в виде списка:

* суммарные статистики: среднее, медиана, дисперсия, энтропия и т.д.;
* временные признаки: наклон, характер тренда, точки пересечения некоторых пороговых значений, значения пиков и их позиции, автокорреляционные коэффициенты и т.д.
* частотные признаки: значения пиков спектра, коэффициенты авторегрессии, коэффициенты, например, вейвлет разложений и т.д.
* другие признаки. В общем случае признаковые пространства могут быть достаточно объемными. Однако, как правило, не все признаки важны, поэтому для таких пространств следует проводить отбор признаков. Существуют и готовые решения, где из объемных пространств уже выделены некоторые наиболее значимые подпространства. Среди таких подходов популярным является Catch22. В данном подходе среди порядка 4500 признаков выделено 22, имеющих большую значимость для стандартных тестов.

В пакете SKTime описанные подходы можно найти в модуле sktime.classification.feature\_based. Давайте протестируем 2 классификатора из этого пакета SummaryClassifier, представляющий описание временных рядов суммарными статистиками и Catch22Classifier, представляющий описание временного ряда указанным выше признаковым пространством.

from sktime.classification.feature\_based import SummaryClassifier, Catch22Classifier  
from sklearn.ensemble import RandomForestClassifier

clf = SummaryClassifier(estimator=RandomForestClassifier(n\_estimators=100))  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(df\_test, y\_test):.3f}')

test score 0.631

clf = Catch22Classifier(estimator=RandomForestClassifier(n\_estimators=100))  
clf.fit(X\_train, y\_train)  
print(f' test score {clf.score(df\_test, y\_test):.3f}')

test score 0.839

Дополнительно отметим, что пакет SKTime также позволяет создавать пайплайны выделения признаков и классификации в ручном режиме. Например, так, как это показано ниже. Показанный пример повторяет алгоритм TimeSeriesForestClassifier, однако, не полностью.

from sktime.transformations.panel.summarize import RandomIntervalFeatureExtractor  
from sktime.utils.slope\_and\_trend import \_slope  
from sklearn.pipeline import Pipeline  
  
steps = [(  
 "extract", RandomIntervalFeatureExtractor(  
 n\_intervals="sqrt", features=[np.mean, np.std, \_slope]  
 )),  
 ("clf", RandomForestClassifier()),]  
  
clf = Pipeline(steps)  
clf.fit(df\_train, y\_train)  
print(f' test score {clf.score(df\_test, y\_test):.3f}')

test score 0.910

Также важно отметить возможность работы с некоторыми другими библиотеками работы с признаками. Так, одной из наиболее популярных библиотек является [tsfresh](https://tsfresh.readthedocs.io/en/latest/text/introduction.html). Данная библиотека является достаточно популярной, хотя и не самой современной. На данный момент ведется интеграция функционала tsfresh в SKTime. По этому мы покажем возможность работы с tsfresh как с отдельным инструментом.

В самом простом случае tsfresh требует формат данных в виде DataFrame, в котором все сегменты данных были бы представлены в одной колонке (для однопеременных данных). Для преобразования форматов воспользуемся встроенной функцией sktime from\_nested\_to\_long.

Для выделения признаков tsfresh используется метод extract\_features.

from tsfresh import extract\_features  
from sklearn.preprocessing import MinMaxScaler  
from sktime.datatypes.\_panel.\_convert import from\_nested\_to\_long

Xt\_train = extract\_features(  
 from\_nested\_to\_long(df\_train),  
 column\_id="index",  
 column\_value="value",  
 column\_kind="column",  
 column\_sort="time\_index", )  
Xt\_train.head(2)

Feature Extraction: 100%|██████████████████████████████████████████████████████████████| 10/10 [00:42<00:00, 4.24s/it]

[2 rows x 789 columns]

Xt\_test = extract\_features(  
 from\_nested\_to\_long(df\_test),  
 column\_id="index",  
 column\_value="value",  
 column\_kind="column",  
 column\_sort="time\_index", )  
Xt\_test.head(2)

Feature Extraction: 100%|██████████████████████████████████████████████████████████████| 10/10 [00:23<00:00, 2.38s/it]

[2 rows x 789 columns]

И так мы выделили 789 признаков. Среди признаков есть значения достаточно разных порядков, а также отсутствующие значения. Проведем стандартизацию полученных признаков.

Xt\_train = (Xt\_train - Xt\_train.min())/(Xt\_train.max() - Xt\_train.min()).astype(np.float32)  
Xt\_test = (Xt\_test - Xt\_train.min())/(Xt\_train.max() - Xt\_train.min()).astype(np.float32)

Xt\_train = Xt\_train.dropna(axis=1)  
Xt\_test = Xt\_test[Xt\_train.columns]

clf = RandomForestClassifier(n\_estimators=100)  
clf.fit(Xt\_train, y\_train)  
clf.score(Xt\_test, y\_test)

0.5798165137614679

Отметим, что при необходимости в пакете SKTime можно найти и ряд не рассмотренных тут преобразований, техник выделения признаков и техник классификации временных рядов.

# Юнит 25 – Когда нужно использовать глубокое обучение нейронных сетей

**Классические модели, управляемые данными (или подход машинного обучения)** . Такие методы, как опорные вектора (SVM), случайный лес, XGBoost, TSF, ROCKET, и тд.

* + Позволяют работать с сильно нелинейными данными.
  + Нет необходимости в статистической гипотезе для модели.
  + Хорошо справляется с нестационарными отношениями между данными.
  + Легко тренировать.
  + Иногда но всегда и не полностью результаты интерпретируются.

**Но**

* + Точность сильно зависит от выбора значений гиперпараметров.
  + Неявная зависимость выбранной модели и данных от результатов прогноза.
  + Точность сильно зависит от сходства между обученными данными и данными вывода.
  + Трудно достичь сопоставимой точности с подходом на основе модели для относительно простых данных.
  + Часто нужно использовать ансамбль методов для достижения приемлемой точности

Вычислительная сложность работы значительно выше, чем для модельных подходов

**С другой стороны**

**Глубокая нейронная сеть**.

* + - Нет необходимости в статистической гипотезе или конкретной форме модели.
    - Может аппроксимировать любую функцию с пропущенными данными, аномалиями и другими нерегулярными шаблонами.
    - Позволяют работать с огромными многомерными рядами данных со сложной взаимосвязью поведения между данными.
    - Автоматически извлекают и обрабатывают сложные признаки и отношения между ними.
    - Для сложных и многомерных данных в большом количестве модели, управляемые данными, могут обеспечить лучшую производительность.

**НО**

* + - Требуется тщательная настройка гиперпараметров.
    - Часто требуется ансамбль сетей для получения высокой точности.
    - Тяжело перетренировать.
    - Трудно достичь сопоставимой точности с подходом на основе модели для относительно простых рядов.

Выбор конкретных методов зависит от поставленной задачи.

Для простых и одномерных данных работает хуже остальных методов.

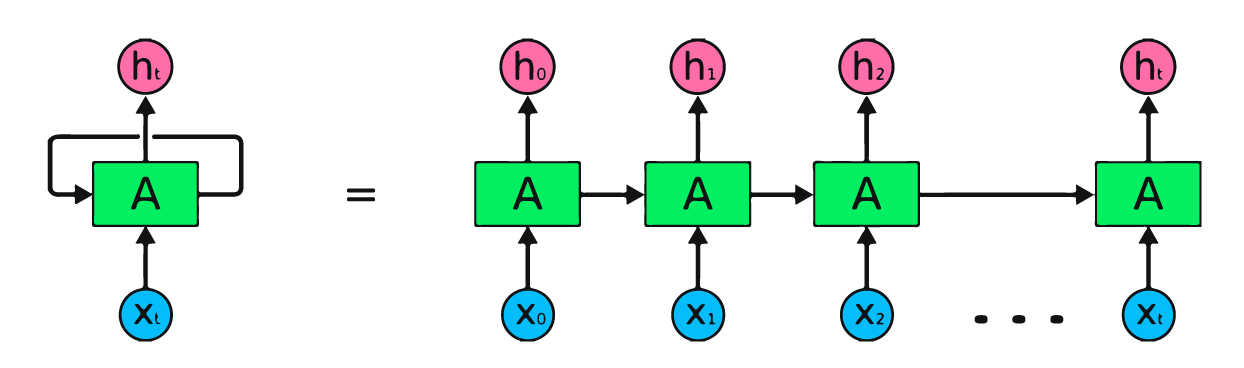
# Юнит 26 – рекуррентные ячейки

Давайте протестируем рекуррентные нейронные сети.

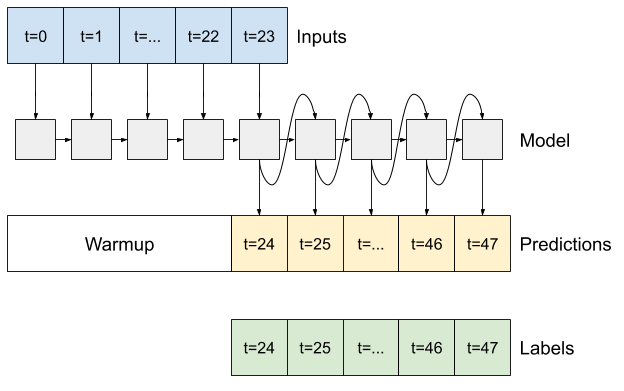
Напомним, что рекуррентные нейронные сети — это тип сетей, основанный на переиспользовании одних и тех же весовых параметров для разных участков данных. При этом сеть как бы скользит по данным.

Отличие рекуррентной сети от обычно полносвязаной сети в использовании скрытого состояния, которое передается из одной позиции окна в другую. Иллюстрация этого подхода приведена на рисунке ниже.

Структура каждой ячейки показана на изображении ниже



Сеть как бы двигается по ВР с заданным окном, но в отличии от полносвязаной сети, результат каждого движения сохраняется и используется в следующий момент. То есть это как бы появляется концепт «память». Другими словами, результат предыдущего положения ячейки сохраняется и используется как некоторое начальное состояние нового положения ячейки. Это состояние мы называем скрытым состоянием.



Таким образом ячейку можно выразить как

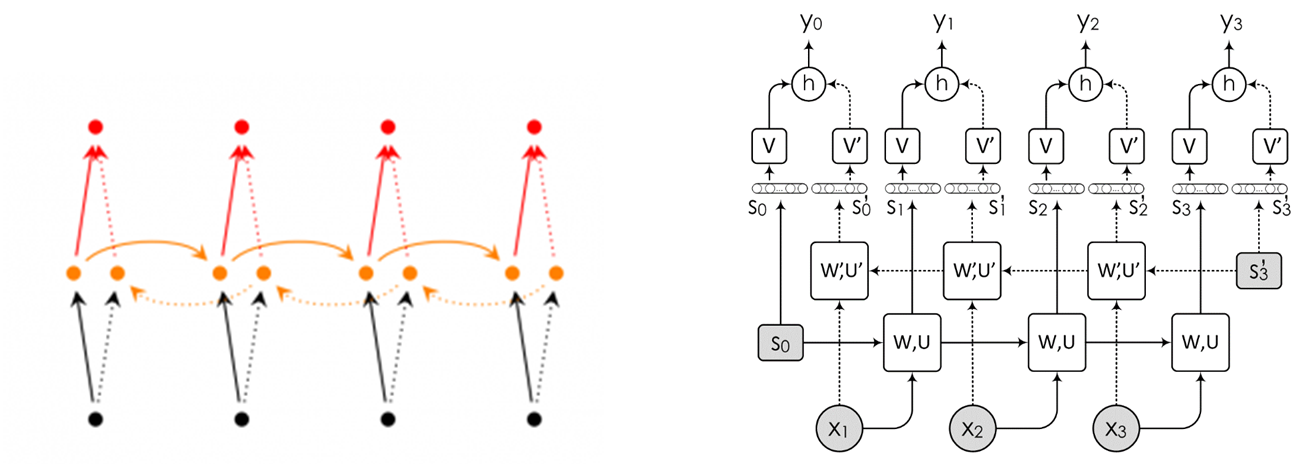
где скрытое состояние в момент ; скрытое состояние в момент ; результат работы ячейки; – входное значение (например ; , – матицы весовых коэффициентов, включая смещение; - функции активации.

Заметим, что если и в каждый момент было бы независимо, то мы перешли к просто полносвязаной сети, то за счет дополнительного уравнения расчета через мы и получаем «память».

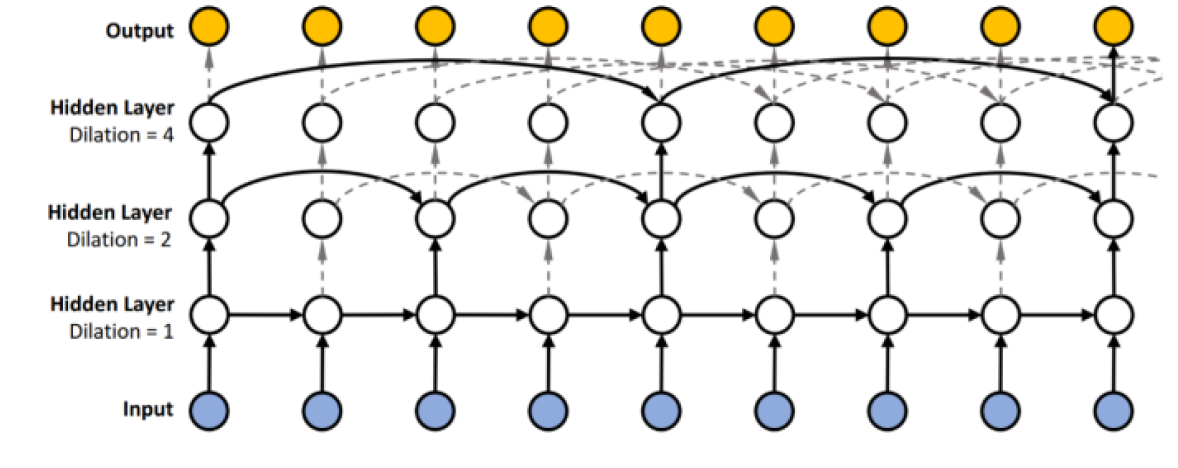
При этом, рекуррентная сеть позволяет сэкономить число параметров используя одни и те же параметры для разных участков ряда. Однако, при этом может возникнуть ряд недостатков. Из-за частого пери использования одних и тех же параметров повышается вероятность переобучения нейронных сетей.

Для компенсанции этого эффекта используют несколько приемов.

* Усечение обучение - каждая эпоха тренировки обучается не на всей последовательности данных, а только в рамках заданной ее длины. Другими словами мы как бы делим на части ВР и каждую часть используем как своего рода независимый пакет.
* Регуляризация.
* Двунаправленное обучение. Два слоя ячеек обучаются независимо давать результат при том что первый слой движется по ВР сначала и до конца, а другой с конца в начало.



* Расширенное обучение, когда используется несколько слоев и в каждом следующем мы равномерно прореживаем выборку в два раза.



* Использование специальных архитектур, таких как LSTM ячейки и GRU ячейки. Про них упомянем позже.

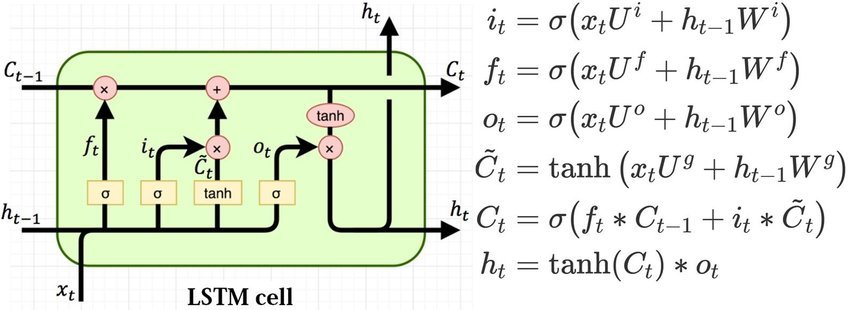
Все до последнего из указанных подходов универсальны.

Также отметим, что следствием высокой вероятности переобучения является не возможность сделать рекуррентные сети слишком глубокими. Архитектуры, содержащие более 4 двунаправленных слоев это редкость.

# Юнит 27 – Про долговременную память.

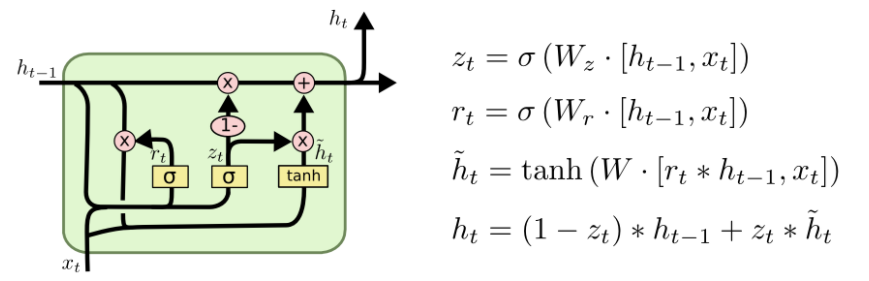
Недостатком усечённого обучения является не возможность учета долговременного контекста. Для решения этой проблемы используются особые рекуррентные LSTM ячейки и GRU ячейки, которые включают учет как коротко-временного, так и долговременного контекста.

Обозначим особенности LSTM ячейки и GRU ячейки. Это два варианта модифицированной RNN ячейки. Отметим, что в литературе можно найти и другие варианты модификаций. Особенностью LSTM ячейки является раздельный учет двух скрытых состояний. Первое из них соответствует кратковременному контексту (например, так мы можем учесть сезонность), а второе состояние учитывает долговременный контекст (так мы сможем учесть тренд). Долговременный контекст – это контекст с долгим затуханием



В ячейке LSTM – это долговременная память, на нее все текущие изменения оказывают сравнительно не большое влияние. В тоже время кратковременный контекст меняется постоянно.

Идея GRU ячейки объединить оба скрытых состояния вместе и передавать их одним параметром. Фактически некоторой памятью «средней» длины. За счет этого общее число параметров будет меньше по сравнению с LSTM ячейкой. Отметим, что часто в анализе временных рядов слишком уж длинная память и не нужна.



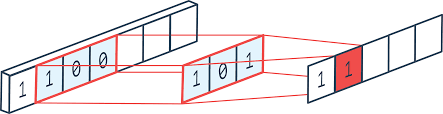
Математически GRU ячейка использует как бы скользящее среднее текущего значения скрытого состояния и своего предыдущего значения.

# Юнит 28 – Сверточные архитектуры.

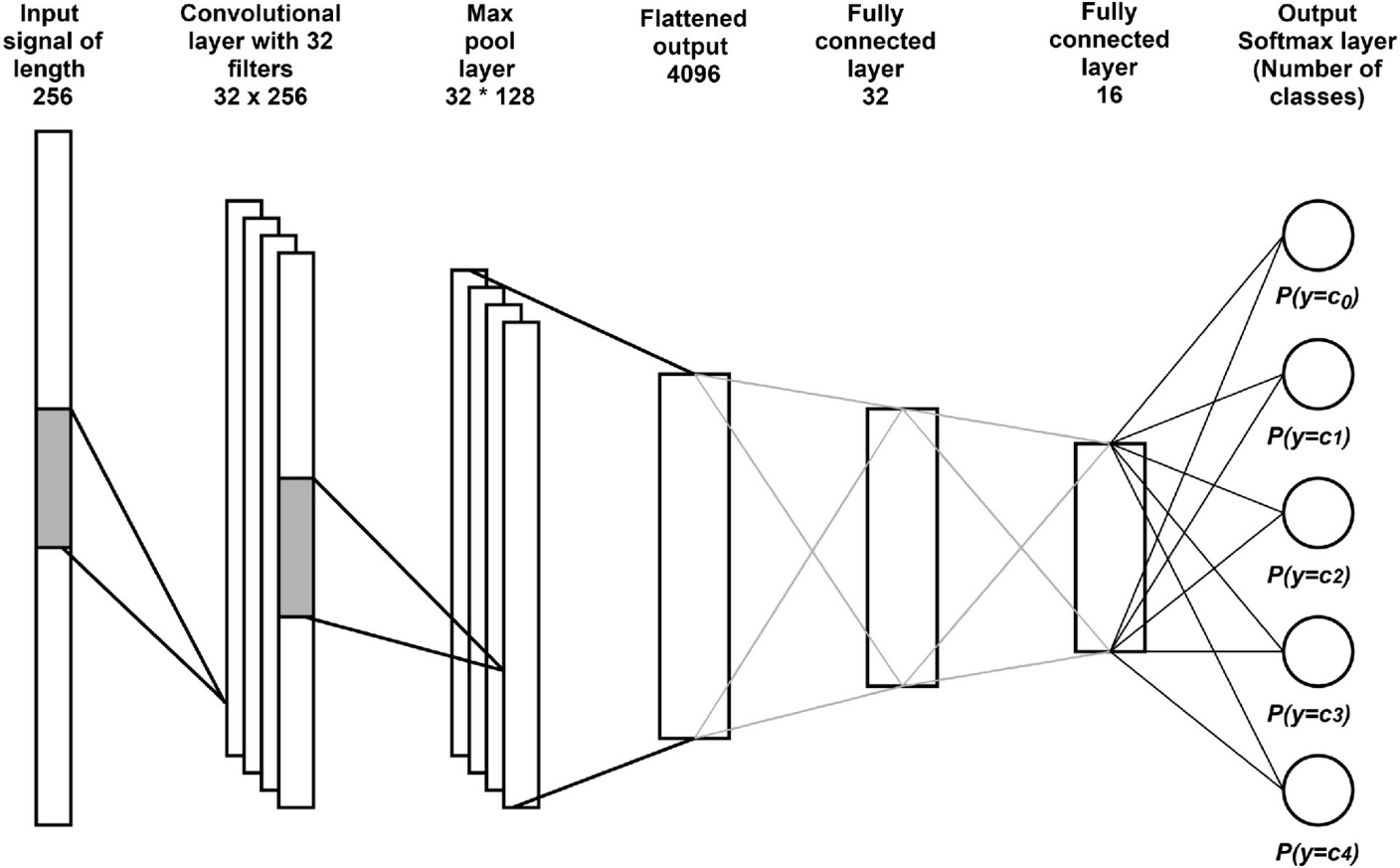
Самый популярный подход в применении нейронных сетей к классификации временных рядов - ​​это одномерные сверточные сети. В 1d свертку можно описать как

где: - вектор весов, , это размер ядра свертки; это так называемая матрица свертки или запаздывания входных данных (лаговая матрица) размер окна данных,

Пример 1-мерной свертки



Как правило из таких сверток можно сформировать архитектуры на подобие уже известных двухмерных. Вот пример одномерного Ленет.

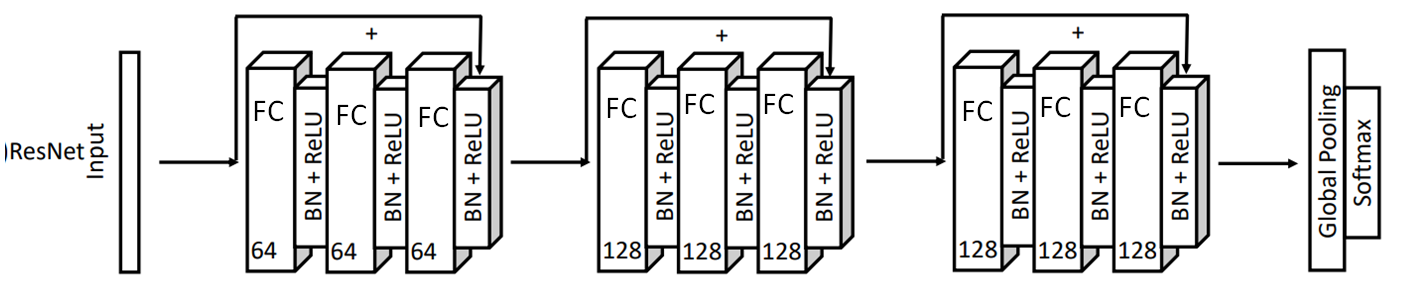


Особенностью одномерных архитектур являются следующие.

* Ядро свертки может быть достаточно большим.
* Если ряд много переменный – это «как бы» многоканальный вход.
* Концепт «рецептивное поле» превращается в длину ряда (контекст) который сеть может охватить для формирования признаков.
* Операция сверточного слоя соответствует «фильтрации» временного ряда. То есть мы ожидаем что результат будет некоторой независимой компонентой ВР. Такая компонента в результате даст признак ВР.
* Помимо сверток в одномерных сетях используются операции пулинга, дропаут, батч-нормализации и прочие, концепты которых активно применяются в анализе изображений.

В некотором смысле, если у нас есть ограниченный сегмент ВР, то это как бы «одномерное изображение». Так полезно думать в задачах классификации ВР.

Аналогично многие из архитектур для анализа изображений транслируются и на классификацию ВР. Вот, например, одномерный **ResNet.**

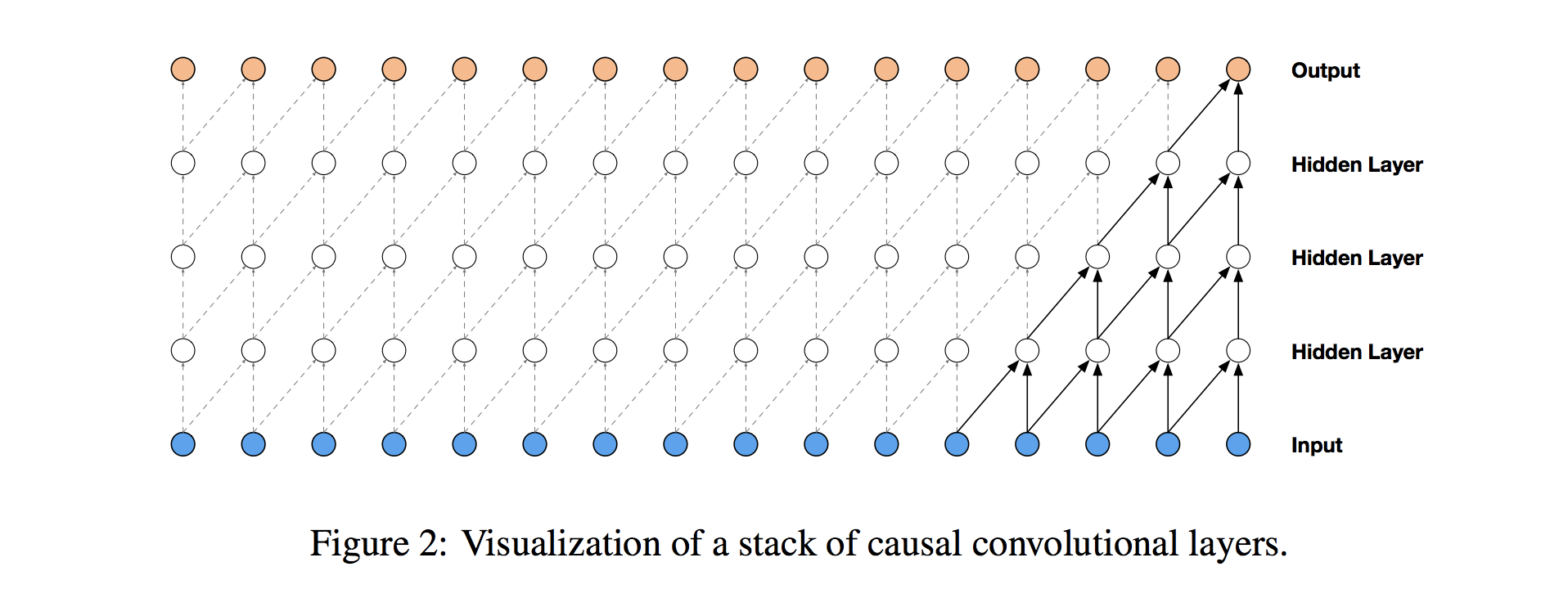


# Юинт 29 – Расширенная свертка

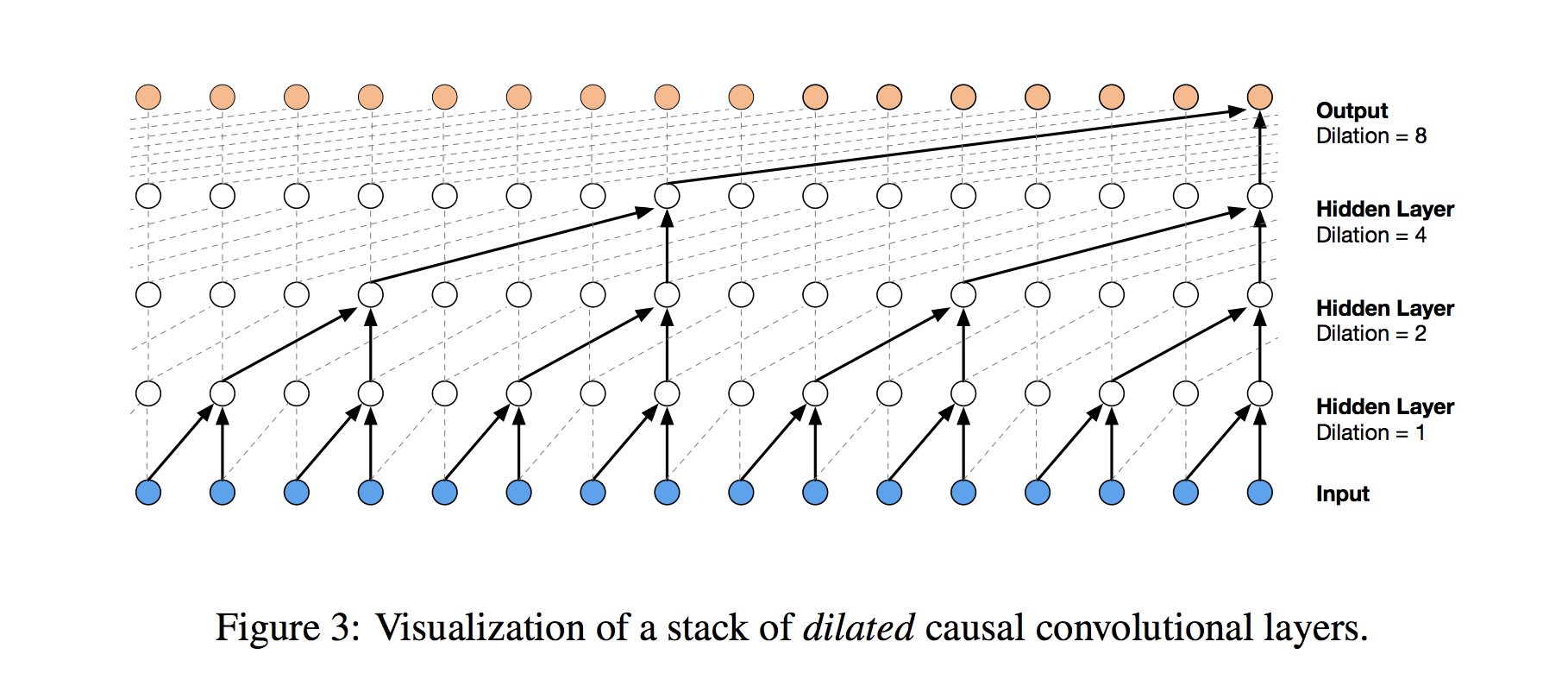
Стандартные 1d-сверточные слои могут быть вычислительно-сложными задачами, если ВР достаточно длинные. То есть если требуется участь весь контекст ВР, то нужно брать очень глубокую сеть. Это в свою очередь усложняет обучение такой сети, увеличивает требования к размеру выборки данных и время работы. Однако, можно избежать этого при использовании концепции расширенной свертки.

*Расширенная свертка позволяет увеличить рецептивное поле (контекст) без нарушения последовательности временного ряда, но за счет увеличения глубины сети.*

В простой каузальной свертке на графике ниже вы можете видеть, что только 5 последних временных шагов могут повлиять на выделенный результат.



Но нужно ли брать все значения для каждого слоя? Не всегда. Допустим что сеть выучит признаки так, чтобы не нарушить теорему Котельникова. Тогда будем брать для свертки не каждое значение, а с прореживанием. Это концепт расширенной свертки. Расширение свертки позволяет *рецептивному* полю экспоненциально увеличиваться в зависимости от глубины сверточного слоя.

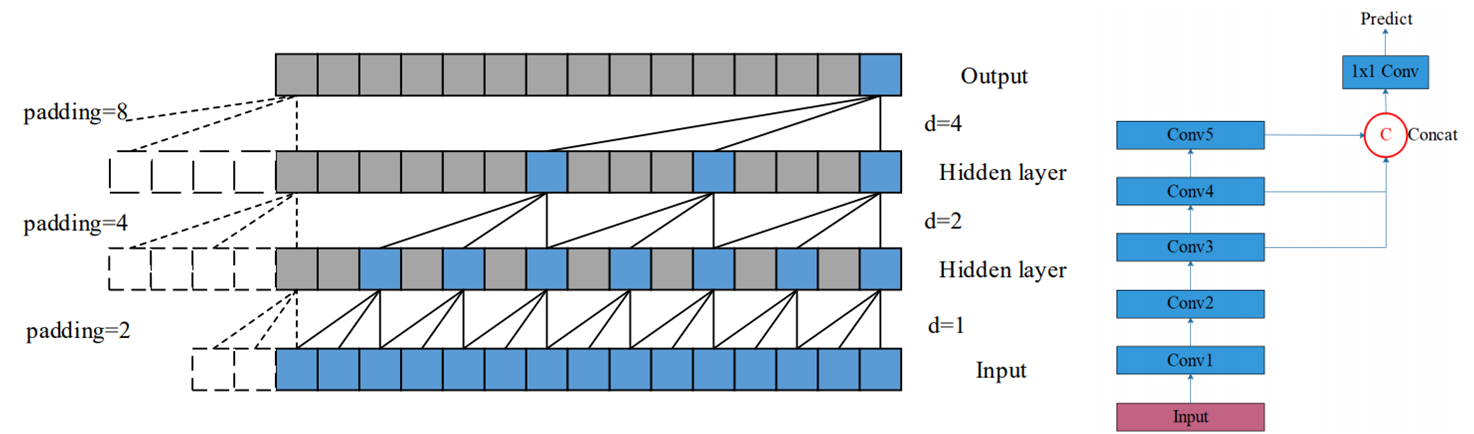


Заметим что в расширенной свертке шаг всегда 1. То есть мы все равно возьмем все значения, но охватим большее рецептивное поле за счет того, что возьмем значения разряжено.

Иногда также используют двунаправленную расширенную свертку.

# 

Также иногда конкатенируют результаты нескольких расширенных слоев вместе, такой подход использован, например, в архитектуре TCN.

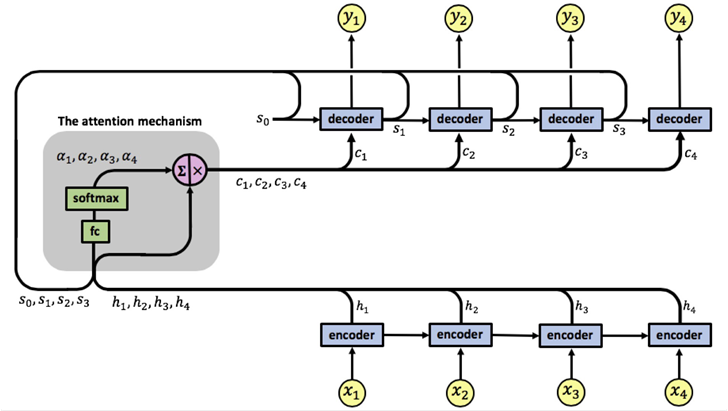


# ~~Юинт 6 – Концепция внимания и трансформеры~~

~~Внимание - механизм улучшения долгосрочного обучения зависимостям в нейронной сети~~

~~Механизм внимания дает несколько ключевых преимуществ – позволяют реагировать на любые значимые, но редкие события. работать как регуляризация, на длинных последовательностях.~~

~~По существу, механизм – это некоторая модификация идеи рекуррентной ячейки. Давайте для предсказания значения использовать не только предыдущие скрытое состояние, но целую группу таких состояний.~~

~~~~

~~На рисунке выше мы фактически мы взяли ряд скрытых состояний, которые соответствуют разным входным значениям. Мы попытались их нелинейно преобразовать в коэффициенты, показывающие влияние каждого из таких входных значений на выходной результат.~~

~~То есть мы учитываем не только последнее входное значение в предсказании результаты, но сразу несколько входных значений. Такой подход должен работать как регуляризация.~~

~~Отметим что для предсказания будущих значений мы можем взять не только наши входные значения, но и результаты предыдущих предсказаний. Однако, это не обязательно. Если учет производится только по входным значениям, то такая модификация называется~~ **~~самовнимание.~~** ~~Это наиболее популярный тип внимания, но не единственный.~~

~~Учет скрытых состояний может производится по-разному.~~

# Юинт 30 - Знакомство с TSAI

Одним из наиболее современных пакетов работы с временными рядами при помощи глубокого обучения нейронных сетей является пакет [**TSAI**](https://github.com/timeseriesAI/tsai), официальная документация по данному пакету может быть найдена [тут](https://timeseriesai.github.io/tsai/).

Пакет tsai — это пакет глубокого обучения с открытым исходным кодом, созданный на основе фреймворков Pytorch и fastai, ориентированный на современные методы для задач временных рядов, таких как классификация, регрессия, прогнозирование, вменение.

Для установки пакета разработчиками рекомендуется следующий скрипт:

stable = True # Set to True for latest pip version or False for main branch in GitHub  
!pip install {"tsai -U" if stable else "git+https://github.com/timeseriesAI/tsai.git"} >> /dev/null

Теперь рассмотрим импорт пакета.

# stable = True # Set to True for latest pip version or False for main branch in GitHub  
# !pip install {"tsai -U" if stable else "git+https://github.com/timeseriesAI/tsai.git"} >> /dev/null

from tsai.all import \*  
import warnings  
  
computer\_setup()

os : macOS-10.16-x86\_64-i386-64bit  
python : 3.9.7  
tsai : 0.3.1  
fastai : 2.7.6  
fastcore : 1.4.5  
torch : 1.10.1  
device : cpu  
cpu cores : 8  
RAM : 16.0 GB  
GPU memory : N/A

from sklearn.model\_selection import train\_test\_split

import torch  
from torch import nn

Теперь давайте выбирем данные. Мы будем использовать тот же пример, что и для классификации в SKTime - набором данных [открытые данные энергетических систем](https://open-power-system-data.org/).

def extract\_country(df\_all, country\_code, year\_min=None, year\_max=None):  
 # List of columns to extract  
 columns = [col for col in df\_all.columns if col.startswith(country\_code)]  
 # Extract columns and remove country codes from column labels  
 columns\_map = {col : col[3:] for col in columns}  
 df\_out = df\_all[columns].rename(columns=columns\_map)  
 # Exclude years outside of specified range, if any  
 if year\_min is not None:  
 df\_out = df\_out[df\_out.index.year >= year\_min]  
 if year\_max is not None:  
 df\_out = df\_out[df\_out.index.year <= year\_max]  
 return df\_out  
  
def transform\_dataframe(df, cols\_map):  
 # Rename columns for convenience  
 df = df[list(cols\_map.keys())].rename(columns=cols\_map)   
 df = df / 1000 # Convert from MW to GW  
 df = df.rename\_axis('Date')  
 return df  
  
def split\_by\_month(df, months):  
 df\_ = pd.DataFrame()  
 for month in np.atleast\_1d(months):  
 df\_ = df\_.append(df.loc[df.index.month == month])  
 return df\_  
  
def to\_segments(df, column, size = 24):   
 df.index.hour[0]  
 start\_idx = 24-df.index.hour[0]  
 df = df.iloc[start\_idx:]  
 val = df[[column]].values  
 return val[:size\*(val.size//size)].reshape(-1,size)

url = 'https://data.open-power-system-data.org/time\_series/2020-10-06/'  
datafile = url + 'time\_series\_60min\_singleindex.csv'  
df\_all = pd.read\_csv(datafile, index\_col='utc\_timestamp', parse\_dates=True, low\_memory=False)  
  
df\_hrly = extract\_country(df\_all, country\_code='DE', year\_min=2015, year\_max=2019)  
  
df\_hrly.sample(3)  
  
cols\_map = {'load\_actual\_entsoe\_transparency' : 'Consumption',  
 'wind\_generation\_actual' : 'Wind',  
 'solar\_generation\_actual' : 'Solar'}  
df = transform\_dataframe(df\_hrly, cols\_map).dropna()  
  
# df wind + solar generation  
df['Wind+Solar'] = df[['Wind', 'Solar']].sum(axis=1, skipna=False)  
df.to\_csv('de\_clf\_data.csv')  
df.head()

Consumption Wind Solar Wind+Solar  
Date   
2015-01-01 07:00:00+00:00 41.133 10.208 0.071 10.279  
2015-01-01 08:00:00+00:00 42.963 10.029 0.773 10.802  
2015-01-01 09:00:00+00:00 45.088 10.550 2.117 12.667  
2015-01-01 10:00:00+00:00 47.013 11.390 3.364 14.754  
2015-01-01 11:00:00+00:00 48.159 12.103 4.198 16.301

winter = to\_segments(split\_by\_month(df, [1,2, 12]), 'Consumption', size = 24)  
summer = to\_segments(split\_by\_month(df, [6, 7, 8]), 'Consumption', size = 24)  
autumn = to\_segments(split\_by\_month(df, [9,10,11]), 'Consumption', size = 24)  
spring = to\_segments(split\_by\_month(df, [3, 4, 5]), 'Consumption', size = 24)  
print(winter.shape, summer.shape, autumn.shape, spring.shape)  
  
X = np.concatenate((  
 winter,   
 summer,   
 autumn,   
 spring))  
y = np.concatenate((  
 1\*np.ones(winter.shape[0]),  
 2\*np.ones(summer.shape[0]),  
 3\*np.ones(autumn.shape[0]),  
 4\*np.ones(spring.shape[0])  
 ))  
  
  
X = np.atleast\_3d(X).transpose(0,2,1)  
y.astype(int)

(448, 24) (458, 24) (452, 24) (458, 24)

array([1, 1, 1, ..., 4, 4, 4])

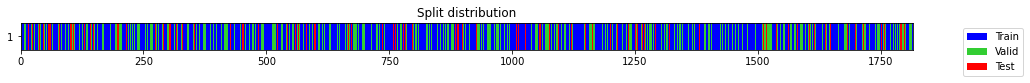
Также давайте сопоставим названия классов и их номера для большего удобстсва.

class\_map = {  
 1:'winter',   
 2:'summer',   
 3:'autumn',   
 4:'spring',   
 }  
class\_map  
  
labeler = ReLabeler(class\_map)  
y = labeler(y)

Особенностью данного фреймворка (как и его основы fastai) является использование супер высокоуровневых API. Это делает фреймворк tsai практически декларативным. Для проведения экспериментом нам потребуется лишь выбор модели и набора данных. Все обучение будет происходить в backend.

Для работы с наборами данных давайте создадим их разделение на тренировочную, валидационную и тестовую выборки. Для этого воспользуемся функцией get\_splits. В рамках урока мы зададим аргумент show\_plot чтобы визуализировать разделение данных.

splits = get\_splits(y,   
 n\_splits=1,   
 valid\_size=0.3,   
 test\_size=0.1,   
 shuffle=True,   
 balance=False,   
 stratify=True,  
 random\_state=42,   
 show\_plot=True,   
 verbose=True)  
splits



((#1091) [1274,358,488,1233,469,725,151,1202,1173,894...],  
 (#544) [1783,1159,952,1712,203,45,563,1071,1640,868...],  
 (#181) [690,310,767,645,80,673,451,597,927,1404...])

Для начала давайте созадим набор данных. Такой набор будет экземпляром класса TSDatasets. В наборе данных зададим разеделние данных и необходимые преобразования tfms.

Также сконфигурируем загручик батчей TSDataLoaders. Загрузим тренировочный и валидационный наборы данных.

tfms = [None, [Categorize()]]  
dsets = TSDatasets(X, y, tfms=tfms, splits=splits)  
   
bs = 256  
dls = TSDataLoaders.from\_dsets(dsets.train, dsets.valid, bs=[bs, bs\*2])

# Юнит 31 – Тестирование ячеек.

Фреймворк tsai позволяет достаточно быстро и без лишних усилий провести анализ различных архитектур применительно к решаемой задаче. Давайте создадим список архитектур, которые можно протестировать.

Среди анализируемых архитектур мы рассмотрим рекуррентные RNN, LSTM и GRU с 3 и 4 слоями, как двунаправленные, так и нет. Кроме рекуррентных нейронных сетей мы также попробуем ряд стандартных сверточных архитектур, таких как ResNet1d, xresnet1d, InceptionTime, XceptionTime и TCN.

Данные архитектуры представляют собой стандартные подходы к классификации временных рядов нейронными сетями. Последняя ахритектура представляет собой использование идеи расширенной свертки. архитектуры ResNet1d и xresnet1d отличаются подходом к организации skip connection. Архитектуры InceptionTime и XceptionTime представляют собой адаптированные под временные ряды соответствующих сверточных сетей для изображений. Также, для сравнения, мы попробуем использовать смешанную архитектуру LSTM\_FCN, полносвязную архитектуру FCN и для сравнения архитектуру трансформер TST.

Отметим, что почти все архитектуры в нашем примере мы вызываем "из коробки", подробней с реализациями данных архитектур можно ознакомится в рамках официальной документации.

archs = [  
 (RNNPlus, {'n\_layers':3, 'bidirectional': True} ),  
 (LSTMPlus,{'n\_layers':3, 'bidirectional': True} ),  
 (GRUPlus, {'n\_layers':3, 'bidirectional': True} ),   
 (RNNPlus, {'n\_layers':4, 'bidirectional': True} ),  
 (RNNPlus, {'n\_layers':4, 'bidirectional': False}),   
 (LSTM, {'n\_layers':3, 'bidirectional': False}),   
 (RNN, {'n\_layers':3, 'bidirectional': True} ),   
 (LSTM, {'n\_layers':3, 'bidirectional': True} ),  
 (GRU, {'n\_layers':3, 'bidirectional': True} ),   
 (ResNet, {}),   
 (xresnet1d34, {}),   
 (xresnet1d50\_deeper, {}),   
 (InceptionTime, {}),   
 (XceptionTime, {}),   
 (TCN, {}),   
 (LSTM\_FCN, {}),   
 (TST, {}),  
 (FCN, {}),   
 ]

Теперь попробуем запустить обучение. Для создания модели используем функцию create\_model, которая для создает модель в формате совместимом с PyTorch для заданной архитектуры и набора гиперпараметров.

В пакете tsai обучение проводится для объекта класса Learner. Таким образом этот класс описывает модель (обученную архитектуру). Мы приведем лишь самую простую конфигурацию этого класса. Конфигурация включает указание загрузчика данных, модели и метрики. В качестве метрики мы используем точность (accuracy). Для запуска обучения в классе Learner используется метод fit\_one\_cycle. Для этого метода в качестве параметров мы зададим число эпох обучения и скорость обучения. Остальные параметры оставим по умолчению. Среди таких параметров будут функция потерь (кросс энтропия) и метод оптимизации ADAM.

По результатам обучения каждой модели мы занесем в таблицу полученные результаты, в том числе точность для валидационного набора данных и время обучения.

from IPython.display import clear\_output  
  
results = pd.DataFrame(columns=['arch', 'hyperparams', 'total params', 'train loss', 'valid loss', 'accuracy', 'time'])  
  
for i, (arch, k) in enumerate(archs):  
   
 model = create\_model(arch, dls=dls, \*\*k)  
   
 print(model.\_\_class\_\_.\_\_name\_\_)  
   
 learn = Learner(dls, model, metrics=accuracy)  
 start = time.time()  
 learn.fit\_one\_cycle(20, 1e-3)  
 elapsed = time.time() - start  
 vals = learn.recorder.values[-1]  
 results.loc[i] = [arch.\_\_name\_\_, k, count\_parameters(model), vals[0], vals[1], vals[2], int(elapsed)]  
 results.sort\_values(by='accuracy', ascending=False, ignore\_index=True, inplace=True)  
 clear\_output()  
 display(results)

arch hyperparams total params \  
0 ResNet {} 478724   
1 xresnet1d34 {} 7229284   
2 InceptionTime {} 388868   
3 xresnet1d50\_deeper {} 21224548   
4 XceptionTime {} 399480   
5 FCN {} 264580   
6 LSTM\_FCN {} 315380   
7 TST {} 411524   
8 RNNPlus {'n\_layers': 4, 'bidirectional': False} 71304   
9 RNNPlus {'n\_layers': 4, 'bidirectional': True} 202604   
10 RNNPlus {'n\_layers': 3, 'bidirectional': True} 142204   
11 RNN {'n\_layers': 3, 'bidirectional': True} 142204   
12 GRU {'n\_layers': 3, 'bidirectional': True} 425004   
13 GRUPlus {'n\_layers': 3, 'bidirectional': True} 425004   
14 LSTMPlus {'n\_layers': 3, 'bidirectional': True} 566404   
15 LSTM {'n\_layers': 3, 'bidirectional': True} 566404   
16 LSTM {'n\_layers': 3, 'bidirectional': False} 203204   
17 TCN {} 66754   
  
 train loss valid loss accuracy time   
0 0.274329 0.150533 0.950368 51   
1 0.190927 0.140133 0.944853 167   
11 0.966967 0.683257 0.729779 16   
12 1.074505 0.892749 0.632353 51   
13 1.077155 0.905379 0.626838 51   
14 1.046785 0.869984 0.608456 68   
15 1.052777 0.883423 0.606618 66   
16 1.129518 0.982424 0.555147 27   
17 1.295903 1.211236 0.547794 94

Среди полученных результатов хотелось бы выделить RNNPlus. Эта архитектура дала результат, оптимальный по соотношению точности и времени работы. Давайте рассмотрим эту архитектуру подробней

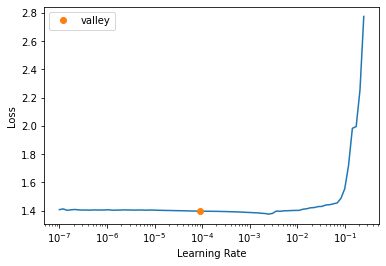
arch = RNNPlus  
k = {'n\_layers': 4, 'bidirectional': True}  
model = create\_model(arch, dls=dls, \*\*k)

model = build\_ts\_model(RNNPlus, arch\_config=k, dls=dls)

Фреймворк tsai предлагает метод определения оптимальной скорости обучения на основе тестов для нескольких эпох обучения. Попробуем определить лучшую скорость обучения.

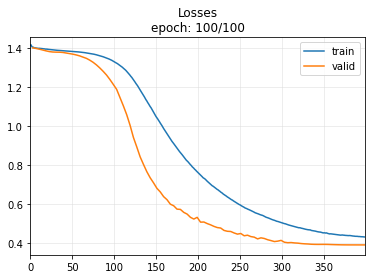
learn = ts\_learner(dls, arch=model, metrics=accuracy)   
learn.lr\_find()

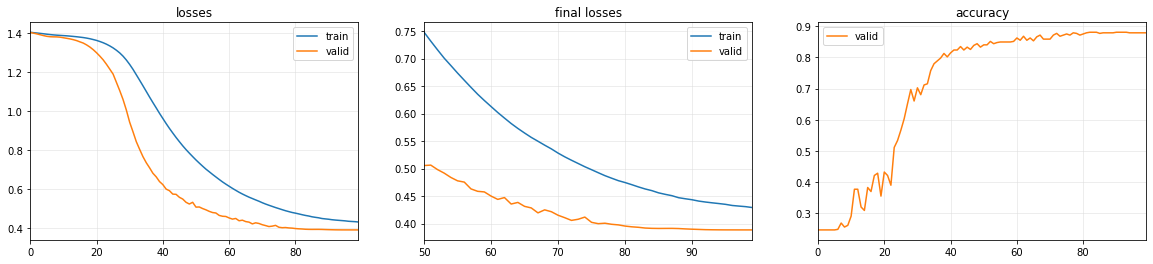
SuggestedLRs(valley=9.120108734350652e-05)



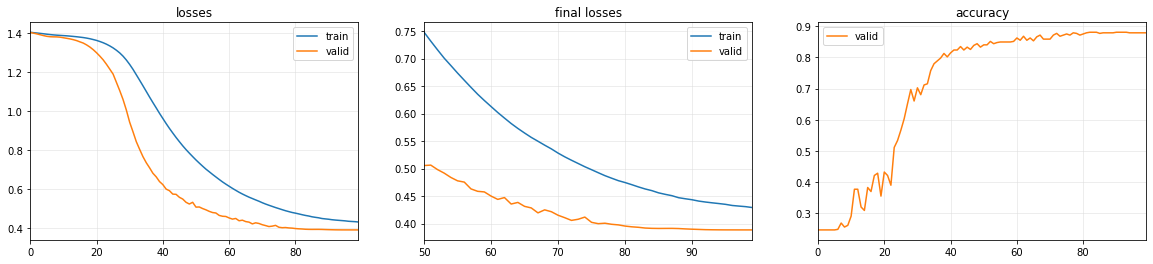
Теперь снова запустим обучение. В этот раз мы запустим процедуру тренировки с коллбеком, позволяющим отслеживать значения функций потерь в режиме он-лайн.

learn = Learner(dls, model, metrics=accuracy)  
start = time.time()  
learn.fit\_one\_cycle(n\_epoch = 100, lr\_max = 0.00009, cbs=ShowGraph())



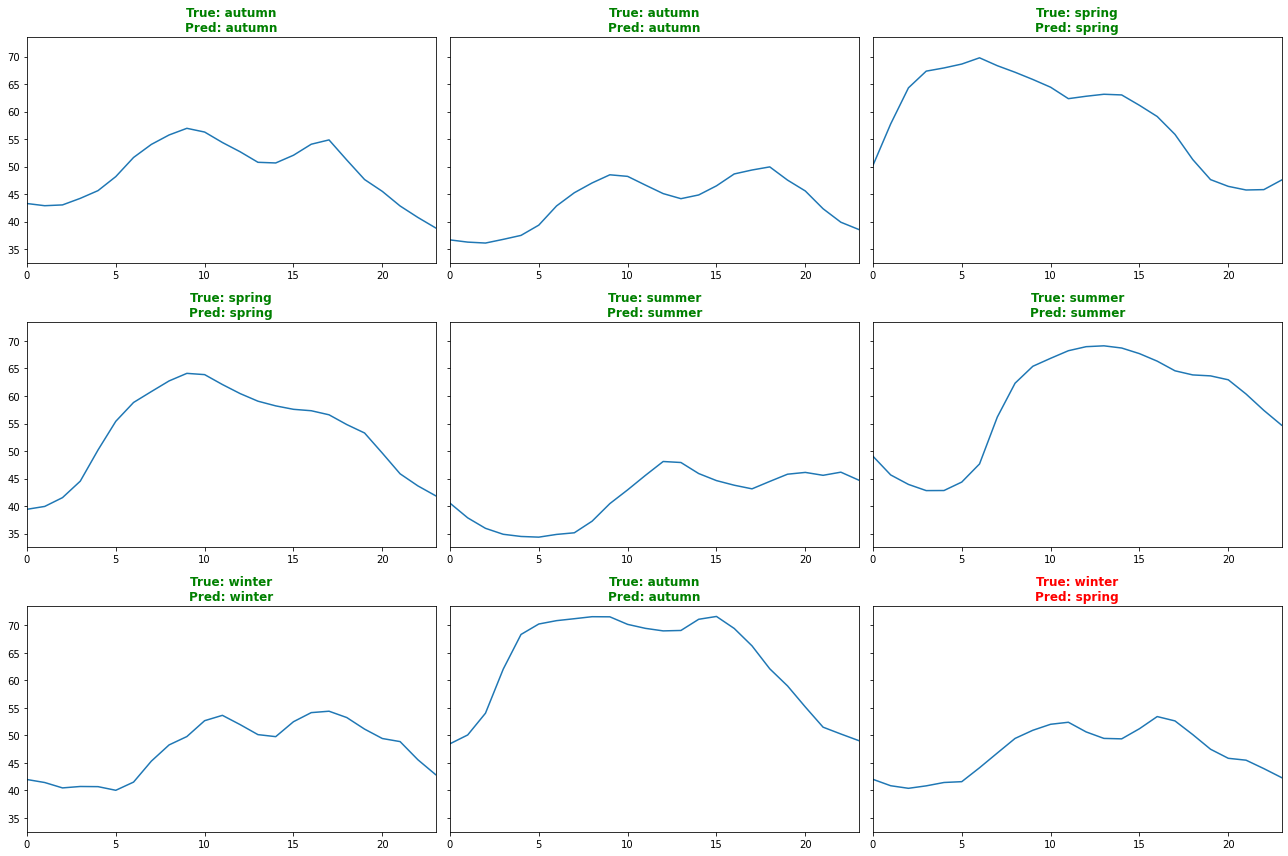


learn.plot\_metrics()



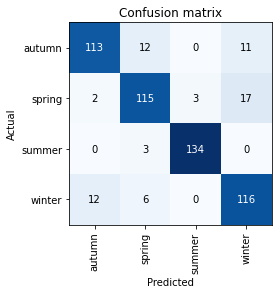
Рассмотрим несколько примеров результатов для валидационного набора данных.

learn.show\_results(sharey=True)



Также для анализа ошибок давайте посмотрим на матрицу ошибок.

interp = ClassificationInterpretation.from\_learner(learn)  
interp.plot\_confusion\_matrix()



Выбирем из матрицы наиболее ошибочные случаи.

interp.most\_confused(min\_val=3)

[('spring', 'winter', 17),  
 ('autumn', 'spring', 12),  
 ('winter', 'autumn', 12),  
 ('autumn', 'winter', 11),  
 ('winter', 'spring', 6),  
 ('spring', 'summer', 3),  
 ('summer', 'spring', 3)]

Помимо прочего поппробуем сделать предсказания для тестового набора данных.

probas, \_, preds = learn.get\_X\_preds(X[splits[2][:5]])  
preds, y[splits[2][:5]]

("['summer', 'winter', 'summer', 'summer', 'winter']",  
 array(['summer', 'winter', 'summer', 'summer', 'winter'], dtype='<U6'))

Оценим точность на тестовом наборе данных

probas, \_, preds = learn.get\_X\_preds(X[splits[2]])

class\_map = {  
 'winter':3,   
 'summer':2,   
 'autumn':0,   
 'spring':1,   
 }  
class\_map  
  
labeler = ReLabeler(class\_map)

from sklearn.metrics import accuracy\_score  
  
accuracy\_score(np.argmax(probas,axis=-1), labeler(y[splits[2]]))

0.8674033149171271

также покажем возможность сохранить и загрузить наборы данных при помощи tsai.

learn.save\_all(path='export', dls\_fname='dls', model\_fname='model', learner\_fname='learner')

learn = load\_learner\_all(path='export', dls\_fname='dls', model\_fname='model', learner\_fname='learner')  
dls = learn.dls

Отдельно выгрузить и загрузить модель для работы (не для тренировки) можно при помощи следующего скрипта.

PATH = Path('./models/RNNn.pkl')  
PATH.parent.mkdir(parents=True, exist\_ok=True)  
learn.export(PATH)

learn = load\_learner(PATH)

probas, \_, preds = learn.get\_X\_preds(X[splits[2][:5]])  
preds, y[splits[2][:5]]

("['summer', 'winter', 'summer', 'summer', 'winter']",  
 array(['summer', 'winter', 'summer', 'summer', 'winter'], dtype='<U6'))

# Юнит 32 - Обнаружение аномалий. Основные определния.

Аномалия - это наблюдение, в котором по крайней мере одна переменная имеет необычное значение в соответствии с заданным критерием.

Проблема обнаружения выбросов или аномалий для временных рядов обычно формулируется как обнаружение резких (относительно кратковременных) изменений значений данных относительно некоторых стандартных, например, предшествующих значений ряда.

Таким образом, части (или сегменты) ряда аномалий должны существенно отличаться от остальных данных в статистическом смысле.

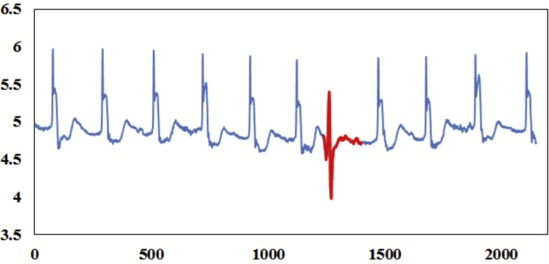
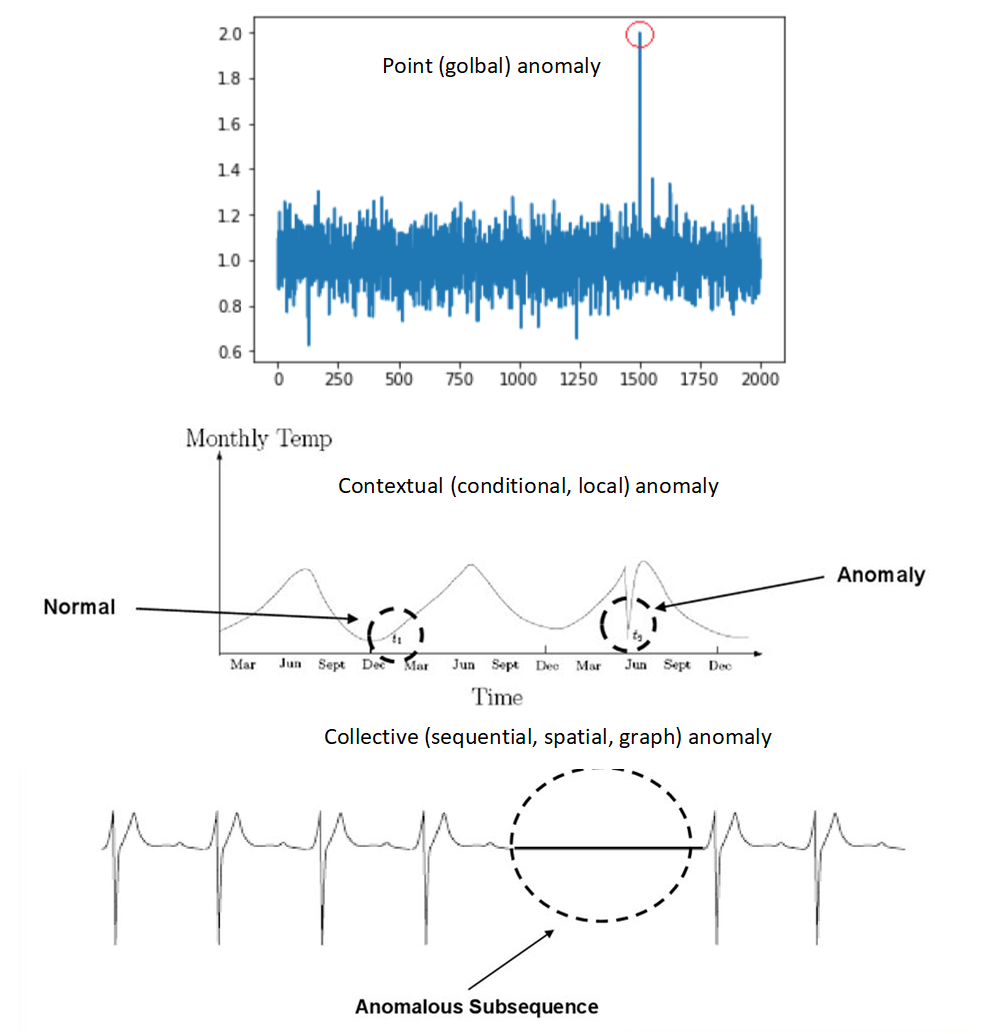


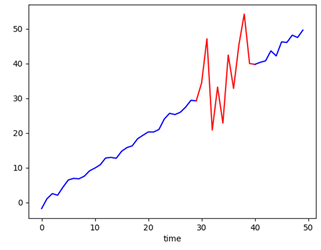
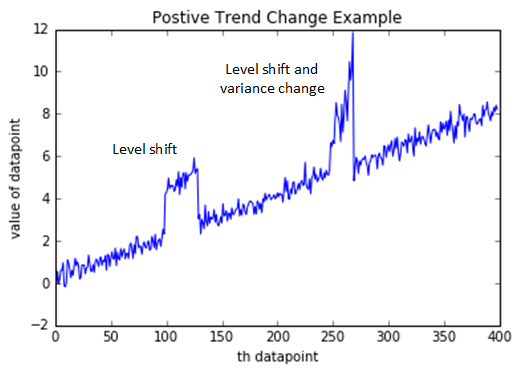
Иллюстрация аномалии.

Во временных рядах можно выделить следующие типы аномалий:

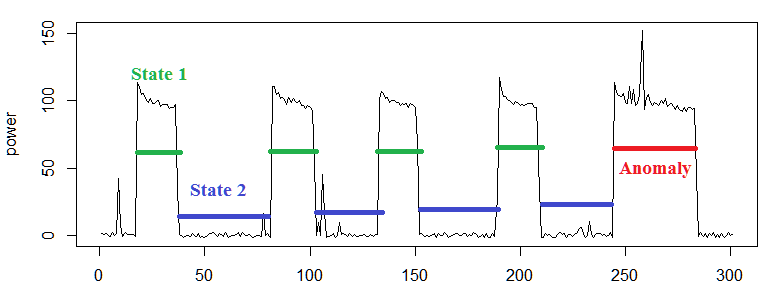
* **точечный выброс** - наблюдение с переменной, имеющей необычное значение.
* **продолжительный выброс (коллективные аномалии)** - это наблюдение, в котором по крайней мере две точки имеют необычные значения.
  + Относительно длительное поведение аномалии иногда называют контекстной аномалией или коллективной аномалией.
* **Контекстные аномалии** - это сегменты данных, «не являются самими собой» в своем окружении, но не глобально.
* **Глобальные аномалии** – это отдельные данные, чье значение значительно превышает общий диапазон значений.



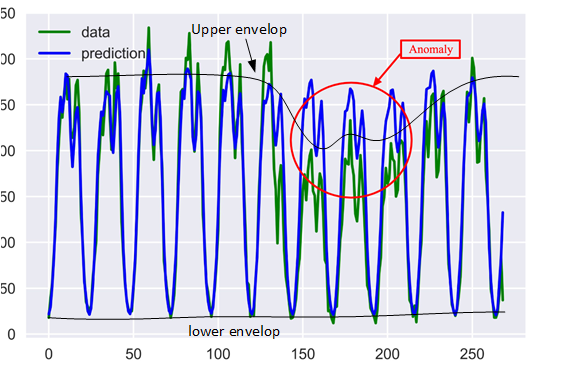
* По  своему поведению можно выделить следующие  **типы аномалий** .
  + Внезапный  **точечный скачок** , отсутствие или коллапс  значения метрики (или выборки).
  + Внезапное и кратковременное  **изменение дисперсии**.
  + Внезапный и кратковременный  **сдвиг уровня**.
  + Внезапное и кратковременное  **изменение огибающей**.
  + Внезапное и  **кратковременное отсутствие паттернов или изменение**  поведения.
  + **Сочетание**  предыдущих случаев.

****

Пример аномалии изменения дисперсии, а также пример аномалии типа сдвиг уровня   
(справа).

****

Пример сезонная аномалия

****

Пример аномалии изменения огибающей

Как правило, если аномалии контекстные, то и правило того, что считать аномалией должно приниматься в зависимости от задачи.

Также важно отметить, что всегда важно понять корневую причину аномалии нежели   
чем просто ее удалять (заменять на пропуск) и т.д.

Примеры корневых причин аномалий могут быть следующие:

* Обнаружение новизны в данных/ точки смены поведения;
* Выявление отклонений типа выброс (сбои, пропуски значений);
* Обнаружение неисправностей  
   (диагностика неисправностей);
* Обнаружение мошенничества или злоупотреблений

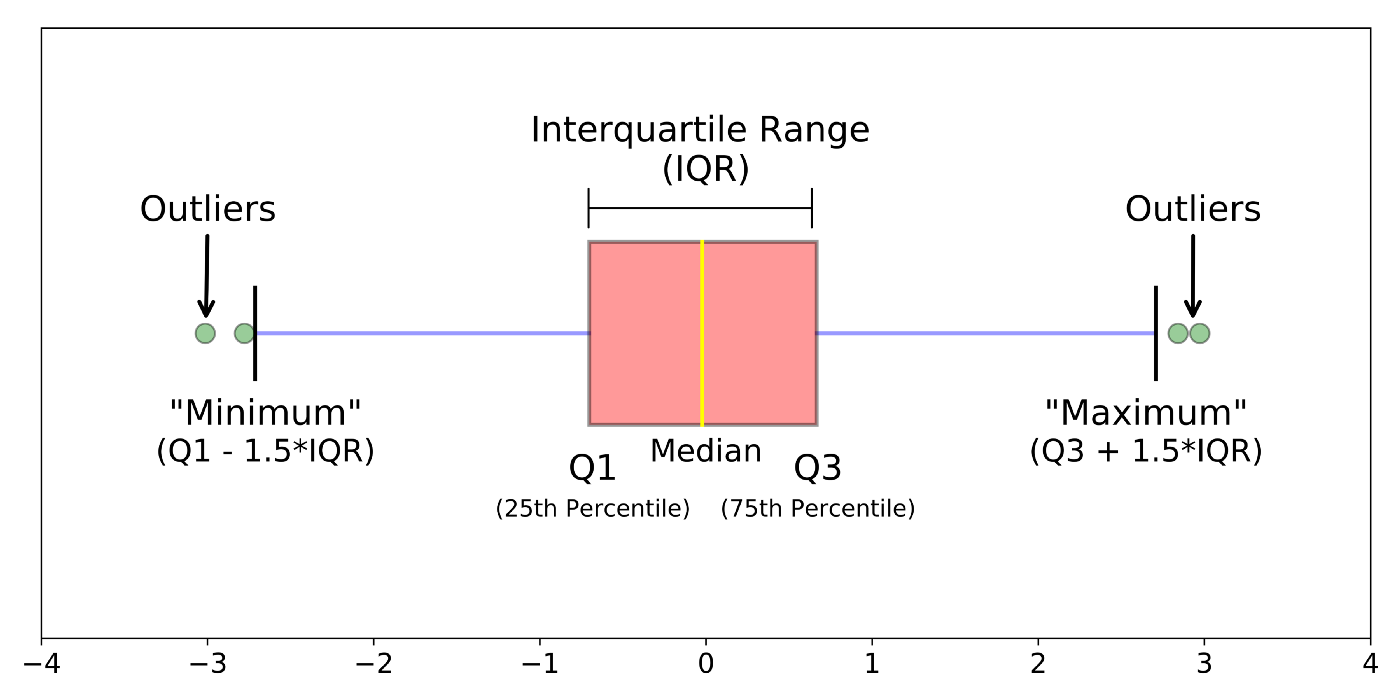
Обнаружение аномалий может быть выполнено с использованием: следующих подходов. Которые по сути представляют собой подходы на основе машинного обучения.

* контролируемые методы обнаружения аномалий. Например, когда вводится два класса, обозначенные как нормальный и аномальный,
* полу-контролируемые методы обнаружения аномалий, когда известен  
  один класс, помеченный как нормальный, и есть множество немаркированных данных,
* неконтролируемые методы обнаружения аномалий, когда данные не размечены, но есть правило отбора, которому должны удовлетворять нормальные данные. Например, значение нормальных данных не должно быть больше установленного порога.

# Юнит 33 – Некоторые методы обнаружения аномалий во временных рядах.

Во многих случаях в качестве основы для сравнения методов обнаружения аномалий может применяться межквартильный размах (IQR). Метод представляет собой отображение распределения данных в виде 4-квантилей.

Выбросы можно рассматривать как все значения квантилей выше 75% или ниже 25%.



Как правило, IQR графически представлены в виде   диаграммы типа ящик с усами. Теория обнаружения аномалий с использованием IQR заключается в том, что, если точка данных находится слишком далеко от 1-го и 3-го квартилей, это, вероятно, является выбросом. Порог выбросов можно выбрать как рекомендованный  
 или вручную. Отметим, что для нормального распределения указанный интервал соответствует значению 2.698 СКО. Также аномалии можно было выделить как порог 3 СКО (где СКО – это стандартное отклонение.)

Иногда полезно перед выставлением таких пороговых значений для нормированных данных. Данные могут быть стандартизованы как

или более устойчиво к разбросу как

где ’ и это преобразованные и исходные данные; – это среднее значение; – это медиана данных; - это стандартное отклонение – это абсолютный разброс относительно медианы.

Отметим также, что если задача ставится как полу контролируемая, то есть выборка, для которой известно, что в ней нет аномалий, то выражения можно заменить на или более устойчиво к разбросу как , где – это данные класса 0 (без аномалий).

По существу указанные методы сводятся к выставлению некоторых порогов (например порога типа . Порог также может быть выставлен вручную, например как некоторый глобальный порог значений (для обнаружения только глобальных аномалий) или например, выбрав окно в рамках которого будут определяться аномалии (локальное окно).

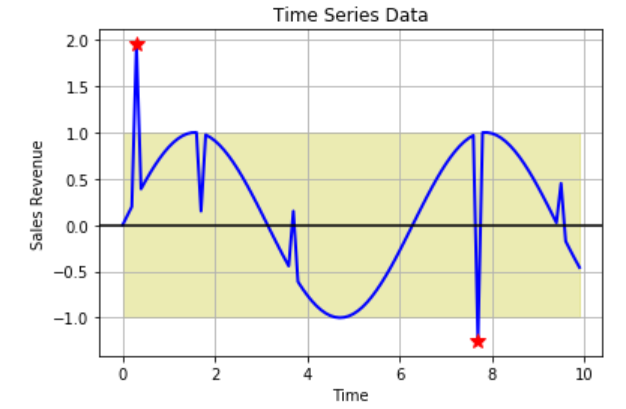
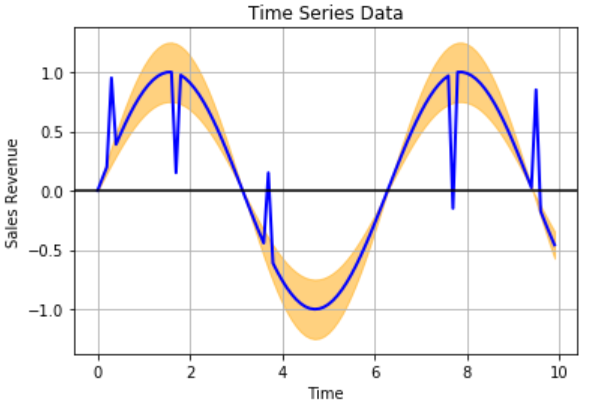
 

Иллюстрация выделения глобальных аномалий по порогу (слева) и локальных аномалий по окну (справа).

Аналогично аномалии могут быть выделаны не только для оригинального ВР, но и для его преобразований или функций от него. Например, по значениям СКО или среднего в движущемся окне.

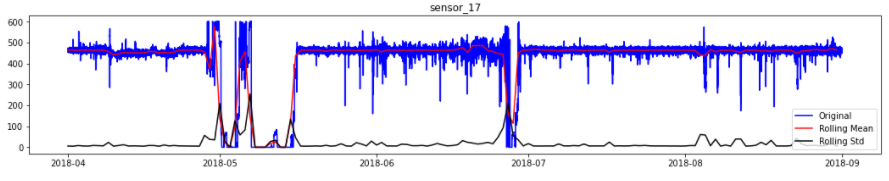


Иллюстрация СКО и среднего для ВР.

Среди таких методов выявления аномалий можно также выделить метод поиска как ошибки предсказаний модели. Пусть мы взяли модель нейронной сети (или авторегрессии) и нашли ее коэффициенты для всего ВР. Теперь пройдемся моделью по имеющемуся ВР и будем смотреть на ошибку предсказаний. Тогда аномалиями будут значения, где ошибка будет выше некоторого порога. Такой порог может быть выбран в абсолютных цифрах или локально. Если известен участок данных без аномалий, то рекомендуется оценивать коэффициент моделей предсказания на нем. Тогда метод становится полу контролируемым.

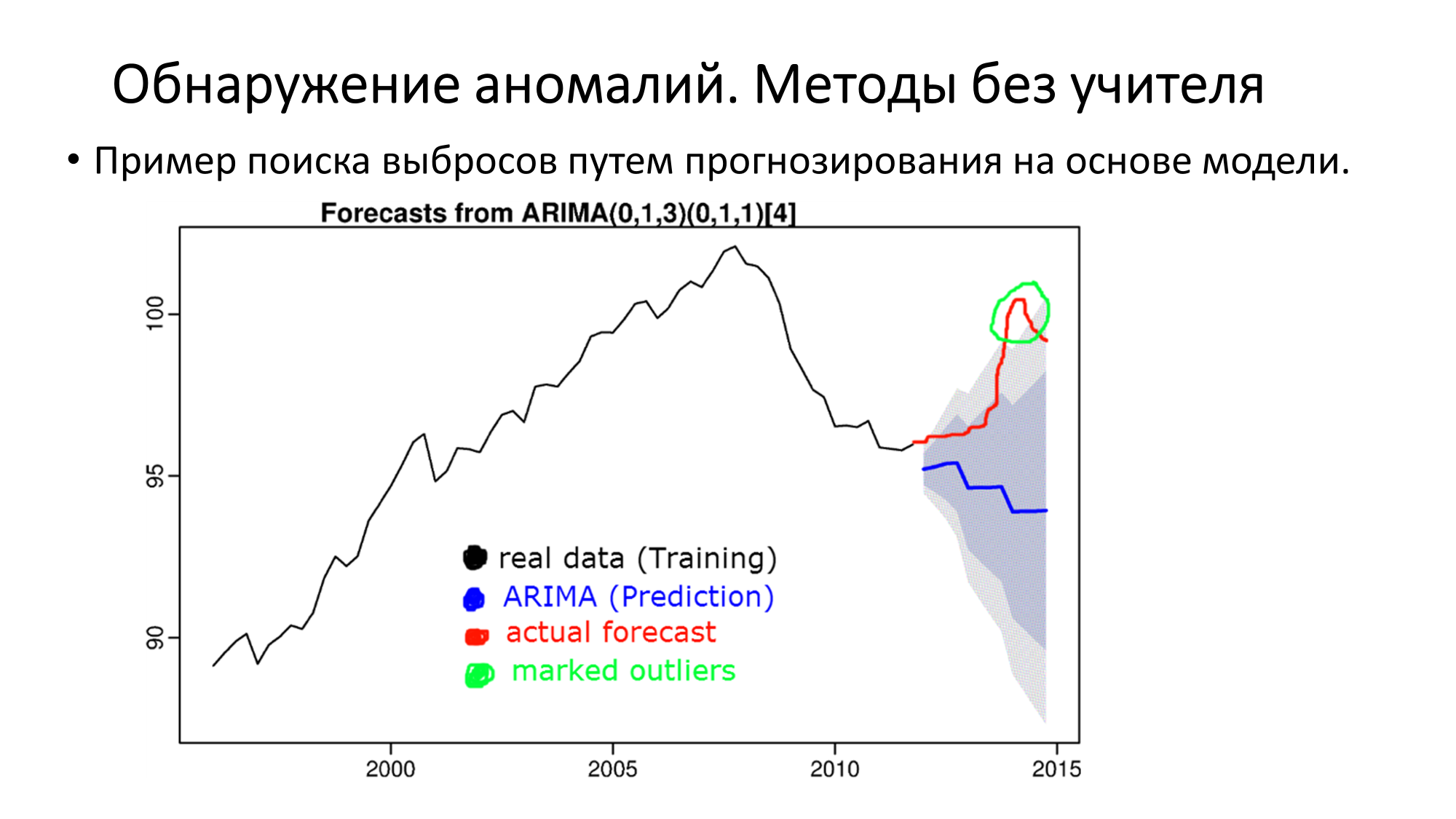
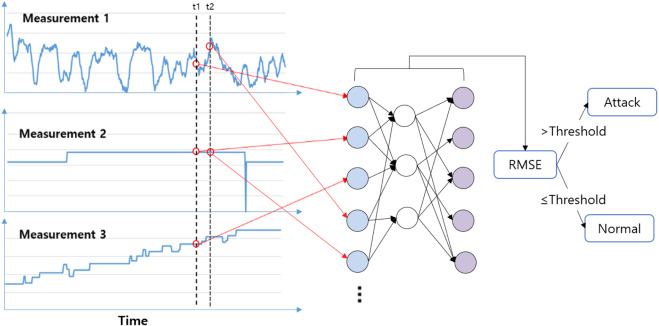


Иллюстрация выявлений аномалий как ошибки работы модели ARIMA.

К такому же классу методов можно отнести задачу восстановления сегментов данных после сжатия. Например, сжатия автоэнкодером. Однако, результаты такого подхода слабо-интерпретируемые.



Пример использовании автоэнкодера для обнаружения аномалий как «плохо» восстановленных данных. Тут мы ожидаем что автоэенкодер не может научиться восстанавливать данные, если они были достаточно редкими в обучающей выборке.

~~Также задача обнаружение аномалий может быть поставлена как задача обнаружения необычных форм (закономерностей, поведения) временного ряда. То есть поставим задачу так:~~

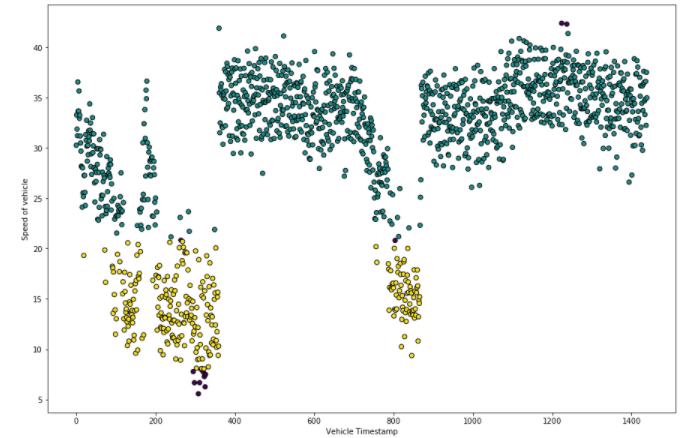
~~Разделим ВР на пересекающиеся сегменты.~~

~~Для каждого сегмента выдели признак, который нас интересует (~~

Другими методами обнаружения аномалий являются, полагаем, уже известные методы из группы классического подхода к временным рядам. Такие методы не учитывают природу ВР. Поэтому такие методы могут выделить только точечные аномалии и не всех типов. Однако, среди методов выделим некоторые, популярные. К таким относится метод кластеризации DBSCAN.

Напомним, что метод DBSCAN основан на предположении, что данные будут разбиты в отдельные кластеры, где кластер с относительно небольшим количеством элементов (или точки вне кластеров) может считаться аномалией.

* При этом кластеры мы строим, задавшись минимальным числом точек и максимальным расстоянием от центроида кластера. Любая точка, которая не является центральной или пограничной, то есть вне кластеров называется точкой шума или выбросом.



Пример использования метода DBSCAN для кластеризации ВР.

Отметим, что помимо указанно, существует целый ряд подходов к обнаружению аномалий, таких, например, как изоляционный лес или одноклассовый метод опорных векторов (для полу-контролируемого обучения). Эти методы также, как и указанный выше метод DBSCAN могут быть использованы для выявления точечных аномалий. Однако, методы не интерпретируемые и не всегда определяют нужные классы аномалий.

Также отметим, что иногда встречается возможность создать две достаточно сбалансированные выборки примеров аномалий и их отсутствия, чтобы говорить о контролируемом обучении. Но такая задача не типична. Если удается выделить две выборки, то задача выявления аномалий сводится к задаче классификации.

# Юнит 34 – практическая работа.

import numpy as np  
import pandas as pd  
  
import matplotlib.pyplot as plt  
from datetime import datetime  
import seaborn as sns  
sns.set(style="whitegrid")  
  
import warnings  
warnings.filterwarnings('ignore')  
  
RANDOM\_SEED = np.random.seed(0)

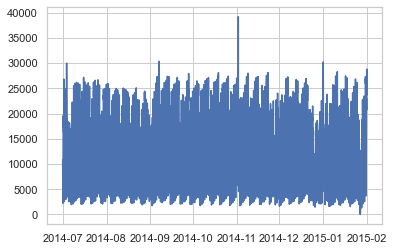
Данные: набор востребованности такси в городе Нью Йорк с шагом в 1 час.

import pandas as pd  
data = pd.read\_csv('https://raw.githubusercontent.com/numenta/NAB/master/data/realKnownCause/nyc\_taxi.csv')  
data['timestamp'] = pd.to\_datetime(data['timestamp'])  
data = data.set\_index(['timestamp']).asfreq('h')  
data.head()

value  
timestamp   
2014-07-01 00:00:00 10844  
2014-07-01 01:00:00 6210  
2014-07-01 02:00:00 3820  
2014-07-01 03:00:00 2369  
2014-07-01 04:00:00 2221

y = data.value.values.reshape(-1, 1)

plt.plot(data.index, y)

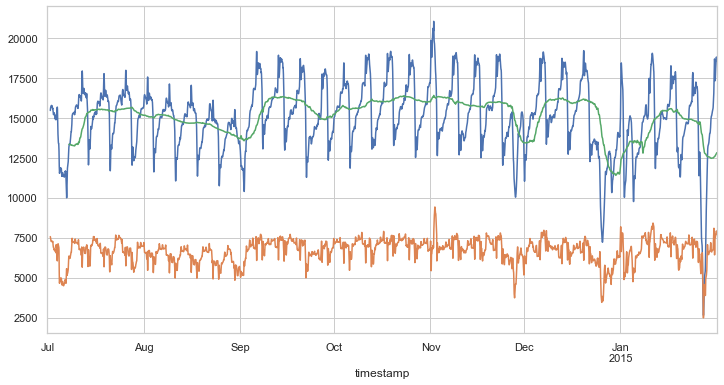


Данные во временных рядах можно рассматривать как в чистом виде, так и преобразо­вать.

Попробуем скользящее средние и скользящие СКО

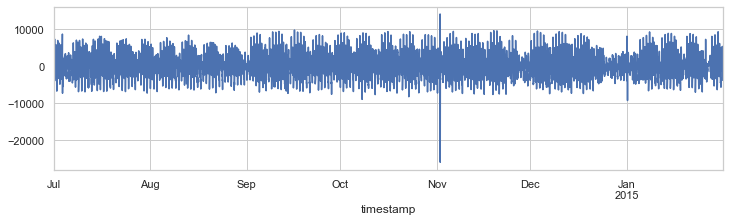
plt.rc('figure',figsize=(12,6))  
plt.rc('font',size=15)  
# create moving-averages  
data['value'].rolling(24).mean().plot()  
data['value'].rolling(24).std().plot()  
data['value'].rolling(24\*7).mean().plot()

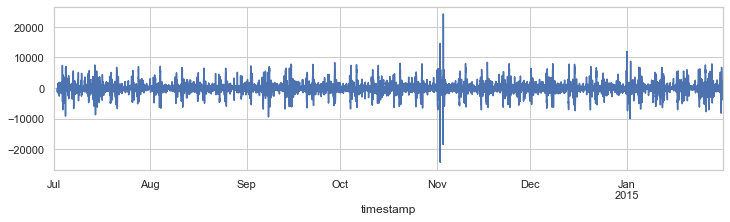
<AxesSubplot:xlabel='timestamp'>



Дифференцирование

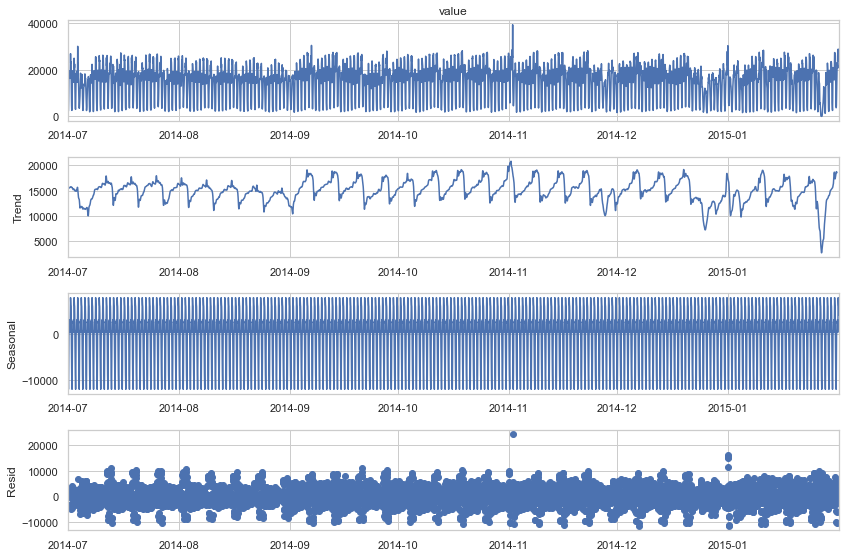
plt.rc('figure',figsize=(12,3))  
plt.rc('font',size=15)  
  
data.value.diff(1).plot(); plt.show()  
data.value.diff(24).diff(1).plot(); plt.show()





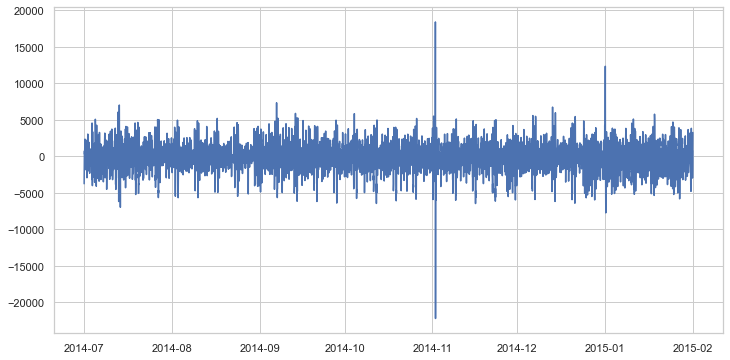
Простое разложение

from statsmodels.tsa.seasonal import seasonal\_decompose  
plt.rc('figure',figsize=(12,8))  
plt.rc('font',size=15)  
  
result = seasonal\_decompose(data.value,model='additive')  
fig = result.plot()



сравнение (остаток, метрика) от предсказания простой моделью

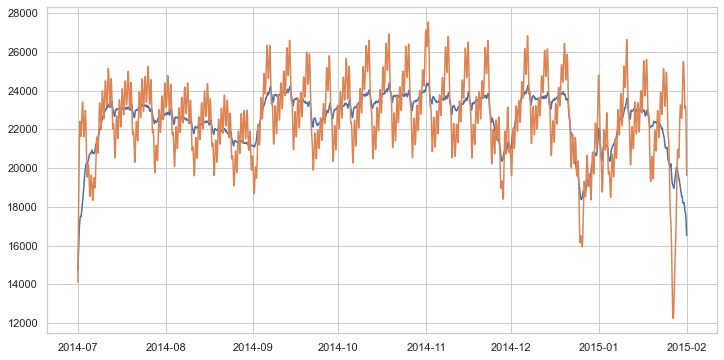
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt  
forecaster = ExponentialSmoothing(y,seasonal\_periods=48,trend="add",seasonal="add").fit()  
predicts = forecaster.predict(start=0, end=len(y)-1)  
  
plt.rc('figure',figsize=(12,6))  
plt.plot(data.index, y.reshape(-1) - predicts)



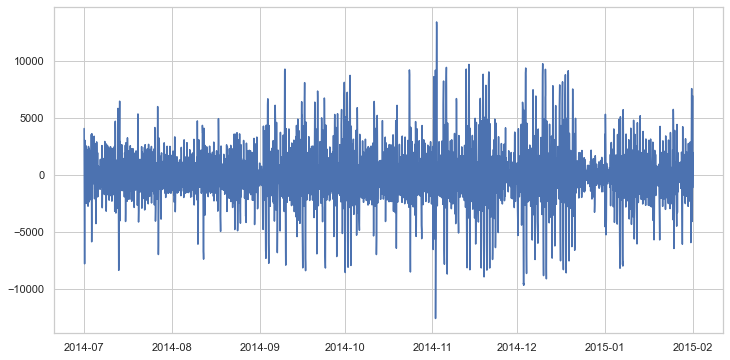
Более тонкие разложения, например по т.н. внутренним модам

import dsatools   
y\_dec = dsatools.decomposition.vmd(y.reshape(-1)[:], order =4)  
plt.plot(data.index[1:-1],y\_dec[0,1:-1])  
  
y\_dec = dsatools.decomposition.vmd(y.reshape(-1)[:], order =2)  
plt.plot(data.index[1:-1],y\_dec[0,1:-1])

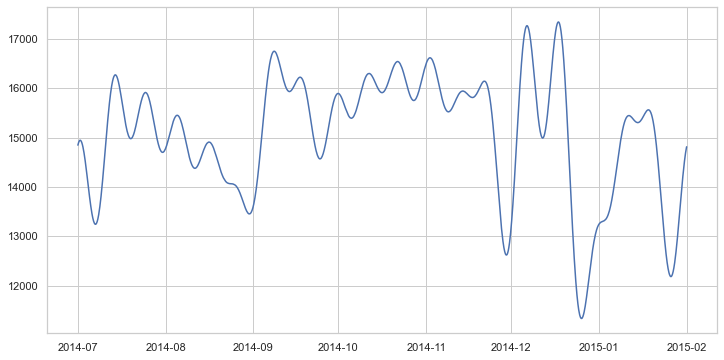
[<matplotlib.lines.Line2D at 0x240926c8da0>]



y\_dec = dsatools.decomposition.emd(y.reshape(-1)[:], order =10)  
plt.plot(data.index[1:-1],y\_dec[0,1:-1]); plt.show()

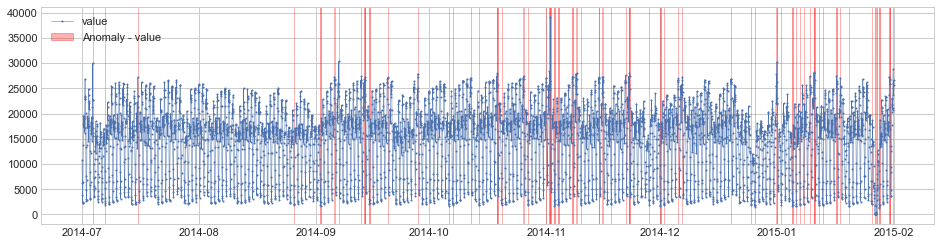


y\_dec = dsatools.decomposition.hvd(y.reshape(-1)[:], order =2)  
plt.plot(data.index[1:-1],y\_dec[0,1:-1]); plt.show()

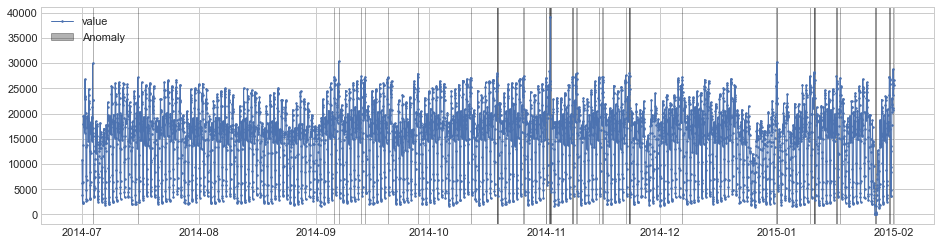


Ряд может быть обработан на основе четких правил

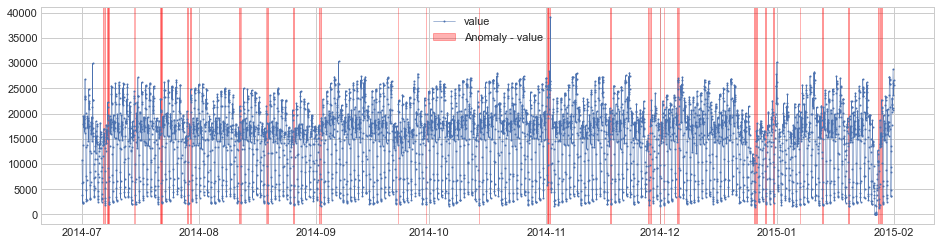
from adtk.detector import QuantileAD  
from adtk.visualization import plot  
quantile\_ad = QuantileAD(high=0.99, low=0.01)  
anomalies = quantile\_ad.fit\_detect(data)  
plot(data, anomaly=anomalies, anomaly\_color='red');



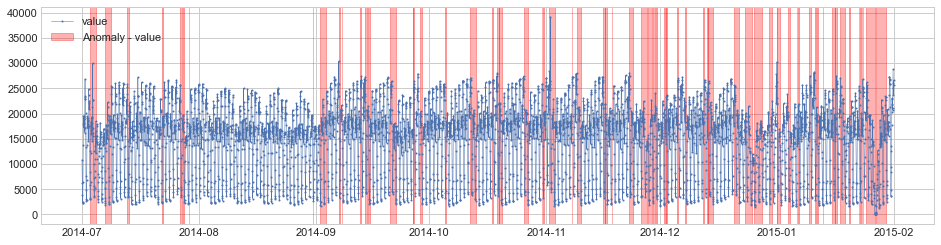
from adtk.detector import ThresholdAD  
threshold\_val = ThresholdAD(high=27000, low=1000)  
anomalies\_thresh = threshold\_val.detect(data.value)  
anomalies\_thresh.value\_counts()  
plot(data, anomaly=anomalies\_thresh, ts\_linewidth=1, ts\_markersize=3, anomaly\_markersize=5, anomaly\_color='black');



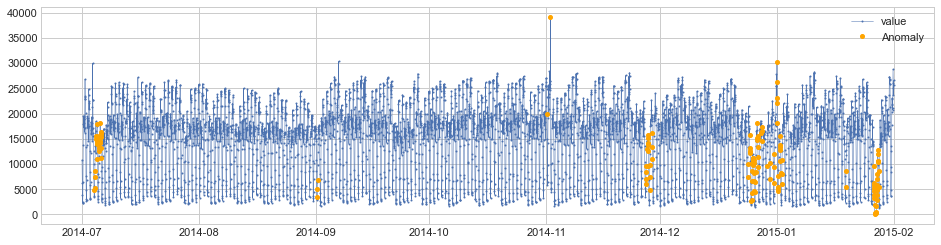
from adtk.detector import VolatilityShiftAD  
volatility\_shift\_ad = VolatilityShiftAD(c=1.0, side='positive', window=30)  
anomalies = volatility\_shift\_ad.fit\_detect(data)  
plot(data, anomaly=anomalies, anomaly\_color='red');



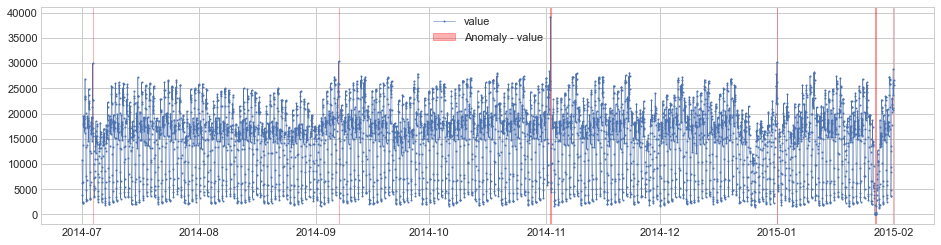
from adtk.detector import LevelShiftAD  
level\_shift\_ad = LevelShiftAD(c=0.2, side='both', window=50)  
anomalies = level\_shift\_ad.fit\_detect(data)  
plot(data, anomaly=anomalies, anomaly\_color='red');



from adtk.detector import SeasonalAD  
seasonal\_vol = SeasonalAD()  
anomalies = seasonal\_vol.fit\_detect(data.value)  
anomalies.value\_counts()  
  
plot(data, anomaly=anomalies, anomaly\_color="orange", anomaly\_tag="marker");

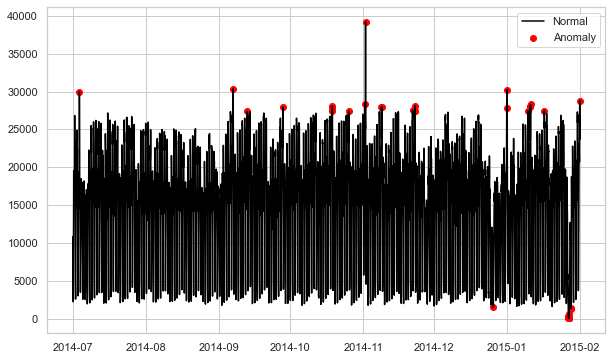


from adtk.detector import InterQuartileRangeAD  
iqr\_ad = InterQuartileRangeAD(c=0.9)  
anomalies = iqr\_ad.fit\_detect(data)  
plot(data, anomaly=anomalies, anomaly\_color='red');

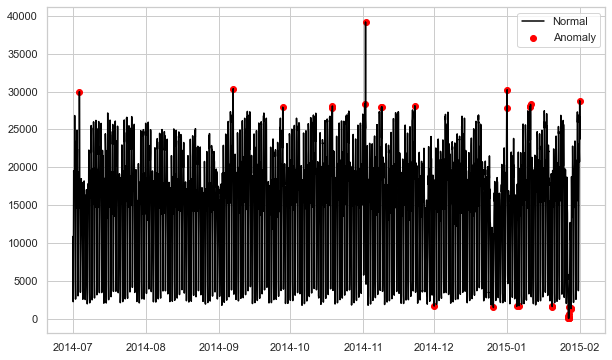


Также могут быть использованы специализированные алгоритмы

from sklearn.ensemble import IsolationForest  
  
outliers\_fraction = 0.007  
  
model = IsolationForest(contamination=outliers\_fraction)  
model.fit(data.value.values.reshape(-1, 1))  
  
data['anomaly\_IF'] = model.predict(y)  
  
# visualization  
fig, ax = plt.subplots(figsize=(10,6))  
  
a = data.loc[data['anomaly\_IF'] == -1, ['value']] #anomaly  
  
ax.plot(data.index, data['value'], color='black', label = 'Normal')  
ax.scatter(a.index,a['value'], color='red', label = 'Anomaly')  
plt.legend()  
plt.show();



from sklearn.neighbors import LocalOutlierFactor  
  
lof = LocalOutlierFactor(novelty=True)  
  
lof.fit(y)  
  
data['anomaly\_LOF'] = lof.predict()  
  
  
fig, ax = plt.subplots(figsize=(10,6))  
  
a = data.loc[data['anomaly\_LOF'] == -1, ['value']] #anomaly  
  
ax.plot(data.index, data['value'], color='black', label = 'Normal')  
ax.scatter(a.index,a['value'], color='red', label = 'Anomaly')  
plt.legend()  
plt.show();



from sklearn.cluster import DBSCAN, KMeans  
model=DBSCAN(eps = 100.)  
# model = KMeans(n\_clusters=10)  
  
model.fit(y)  
colors = model.labels\_  
cls,counts = np.unique(colors, return\_counts=True)  
sort\_idx = np.argsort(counts)#[::-1]  
print(counts[sort\_idx], cls[sort\_idx])  
plt.scatter(data.index, y, c = colors == cls[sort\_idx][-1])

[ 7 9 14 5130] [ 2 1 -1 0]

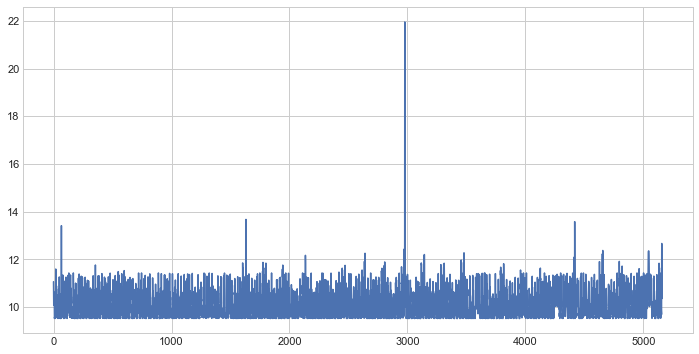
<matplotlib.collections.PathCollection at 0x29567b884e0>



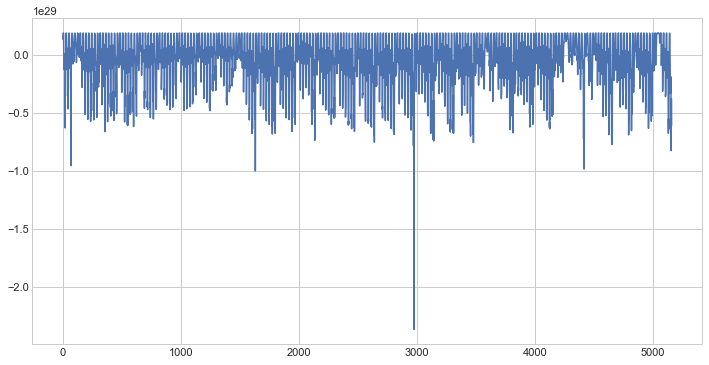
По мимо sklearn есть и другие библиотеки работы с аномалиями, например <https://pyod.readthedocs.io/en/latest/>

import pyod.models  
from pyod.models.gmm import GMM  
  
clf = GMM( n\_components=2, )  
clf.fit(y)  
y\_scores = clf.decision\_scores\_  
plt.plot(y\_scores)

[<matplotlib.lines.Line2D at 0x240adad3978>]

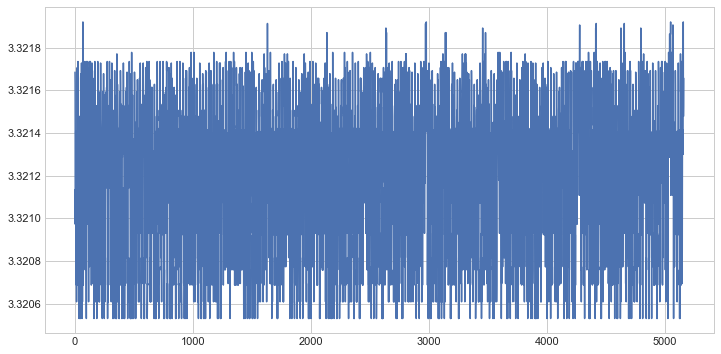


from pyod.models.ocsvm import OCSVM  
pyod.models.anogan.AnoGAN  
clf = OCSVM(kernel='poly',degree=3)  
clf.fit(y)  
y\_scores = clf.decision\_scores\_  
plt.plot(y\_scores)



from pyod.models.hbos import HBOS  
  
clf = HBOS(n\_bins=100, alpha=0.1, tol=0.5, contamination=0.1)  
clf.fit(y)  
y\_scores = clf.decision\_scores\_  
plt.plot(y\_scores)

[<matplotlib.lines.Line2D at 0x240acc81a20>]

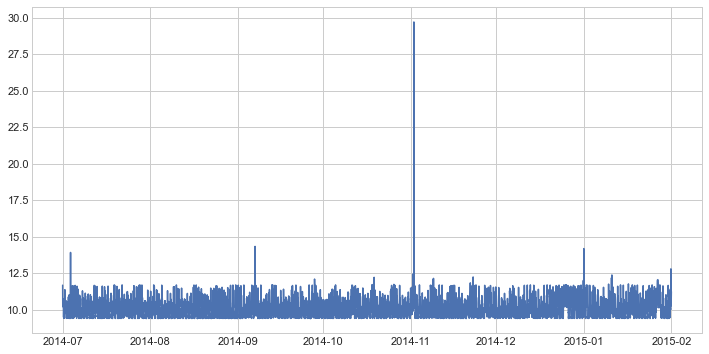


в данной библиотеки pyod можно в т.ч. реализовать попробовать на основе восстановления автоэнокдером

import numpy as np  
import pandas as pd  
from pyod.models.auto\_encoder import AutoEncoder  
  
clf = AutoEncoder(hidden\_neurons =[1,125, 125, 1])  
clf.fit(y)  
y\_scores = clf.decision\_scores\_

Model: "sequential\_1"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 dense\_21 (Dense) (None, 1) 2   
   
 dropout\_18 (Dropout) (None, 1) 0   
   
 dense\_22 (Dense) (None, 1) 2   
   
 dropout\_19 (Dropout) (None, 1) 0   
   
 dense\_23 (Dense) (None, 1) 2   
   
 dropout\_20 (Dropout) (None, 1) 0   
   
 dense\_24 (Dense) (None, 125) 250   
   
 dropout\_21 (Dropout) (None, 125) 0   
   
 dense\_25 (Dense) (None, 125) 15750   
   
 dropout\_22 (Dropout) (None, 125) 0   
   
 dense\_26 (Dense) (None, 1) 126   
   
 dropout\_23 (Dropout) (None, 1) 0   
   
 dense\_27 (Dense) (None, 1) 2   
   
=================================================================  
Total params: 16,134  
Trainable params: 16,134  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

plt.plot(data.index,y\_scores); plt.show()



Заключение:

* Модуль «Использование методов машинного обучения в анализе временных рядов» раскрывает особенности использования методов машинного обучения в анализе временных рядов. В том числе показаны основные особенности данных подходов в отношении этой модальности данных. Напомним, что эти особенности связаны с казуальностью и заданным порядком значений ВР, имеющим четкую интерпретацию. Именно это свойство и дает новую информацию, использование которой позволяет ввести те техники, примеры которых показаны в модуле.
* Рассмотренные в модуле примеры задач в настоящее время имеют высокую степень важности, а методы для их решения актуальны и составляют т.н. State-of-the-art (то есть современный уровень решения). Однако, стоит понимать, что область знаний развивается достаточно быстро. Поэтому специализируясь на данной области нужно всегда поддерживать уровень знаний актуальным.

# Юнит - Словарь

**В ходе изучения модуля будут изучены такие понятия как:**

**Классификация ВР –** задача оценки соответствия некоторых сегментов ВР заданным дискретным категориям – т.н. классам. То есть оценка схожести некоторым множествам примеров. Схожесть может быть разной, например, могут быть следующие варианты:

* + схожесть во временном поведении.
  + схожесть в частотном поведении (наличие определённых компонент в спектре) .
    - А также время-частотное поведение.
  + схожесть по форме (патерн, без привязки ко времени).
    - Схожесть по частоте повторяемости, паттерн – шаблон поведения.
  + схожесть поведения компонет (монотонный тренд, характер сезонности и т.д.).

**Методы на основе сырых данных** подразумевают поиск "схожести" участков ВР для каждого класса как таковых**.** При этом "схожесть" определяется как значение определенной метрики, например, расстояние. Достоинствами данного подхода являются потенциально высокая точность. Однако, для работы подход требует достаточно много вычислительных ресурсов и не всегда позволяет "правильно" учесть всю содержащуюся в рядах информацию. Также отметим, что результаты работы для данного подхода не всегда являются интерпретируемыми, что может представлять отдельную проблему.

**Эластичные меры расстояний** являются модификациями традиционных подходов, устойчивыми к небольшим изменениям в данных. Наиболее универсальным подходом можно считать т.н. Расстояние динамической трансформации (деформации) по времени **(Dynamic Time Warping Distance, DTW).** Алгоритм DTW, как следует из названия, пытается деформировать ось времени одного из сигналов (сжатие/растяжение для каждых двух точек). Деформация производится таким образом, чтобы найти минимальное расстояние между двумя точками. Метод используется, например, в алгоритме **kNN-DTW.** Алгоритм DTW подходит для классификаторов по наличию шаблона.

В противоположность к подходам на полных сегментах данных ("whole series") авторы выделяют **интервальные оценки** ("intervals" ) - то есть оценки по одному или несколькими частям сегмента временного ряда. Могут быть выбраны как одно или совокупность интервалов. Интервалы могут быть выбраны: в соответствии с каким либо критерием, случайно или например на заданной сетке.

**Шейплет** **(Shapelet)** - сравнительно короткий участком (паттерн) временного ряда, который в наибольшей степени позволяет отличить один класс от другого. Шейплеты можно использовать для обнаружения «фазово-независимого локализованного сходства между сериями одного и того же класса». Шейплеты подходят для классификаторов по наличию шаблона или паттерна поведения сегмента.

**Алгоритмы на основе словарей –** это алгоритмы представления ВР или их частей в виде гистограмм некоторого описания в грубых дискретных шагах и квантах значений ВР.

**Подход на основе признаков -** предполагает, что модель строится на выделение признаков и принятии решений по ним. Достоинствами подхода являются низкая избыточная и более высокая интерпретируемость классификатора. Однако не всегда признаковое описание достаточно в рамках решаемой задачи.

**Признаки** - это определенные регулярные характеристиках для каждой последовательности. Признаки:

* + Регулярны для набора данных.
  + Отражают класс (регулярны для класса).
  + Позволяют различать классы как можно более четко

Признаки могут быть выделены как для всего сегмента ВР, так и для интервалов в нем. Могут быть как точечными, так и векторными.

**По группам признаки** можно разделить на:

* Описательные статистики
* Временные параметры
* Спектральные параметры (а также Время-частотные, псевдо спектральные)
* Экзогенные факторы, а также некоторые критерии (например наличие паттерна в данных) также могут быть выбраны как признаки.

Подход на основе признаков предполагает:

* Выделение репрезентативного и не избыточного пространства признаков
* Построение классификатора на их основе

Для многопеременных ВР могут быть несколько подходов к выделению признаков:

* объединение признаков нескольких составляющих в один вектор;
* ансамблирование результатов классификации по каждой составляющей;
* использование специальных методов,

Есть готовые пространства признаков и готовые алгоритмы на их основе. К таковым относят:

* **Лес временных рядов (TSF) –** построение случайного леса по специальному критерию для деревьев, сформированных из временных признаков для случайных интервалов внутри ВР. Canonical Interval Forest (CIF) используетоснову TSF и набора признаков Catch22 с расширением для построения классификатора.
* **Спектральный ансамбль со случайными интервалами (RISE) –** случайный лес, деревья которого представляют признаковое описаниевсего сегмента и интервалов внутри, при этом для каждого участка ВР выделяются векторные признаки имеющие спектральную интерпретацию.
* **И другие.** Часто это набор признаков (например tsfel) и алгоритм на подобие случайного леса или регрессии над ними для принятия решений.

**Комбинации подходов –** использование нескольких гетерогенных подходов к классификации ВР. Используется если одним подходом ВР не описать. методы на основе данных и на основе признаков могут быть объединены в ансамбли таким образом, чтобы учесть преимущества и недостатки тех и других подходов. Пример таких методов **HIVE-COTE.**

**HIVE-COTE -** Метод ансамбля с иерархическим коллективным голосованием на основе преобразований; мета-ансамбль, построенный на классификаторах (The Hierarchical Vote Collective of Transformation-based Ensembles with Collective of Transformation-based Ensembles): шейплеты (паттерны), BOSS (словари – частота паттернов), Лес временных рядов (временное Поведение), RISE (частотное поведение).

**ROCKET** (RandOm Convolutional KErnel Transform) Классификатор - представляющий набор случайных светрточных ядер и признаковое описание над каждым из результатов таких сверток. Как правило ядер достаточно много. Модификации алгоритма типа mini-rocket позволяют упростить модель.

**Задача обнаружения аномалий в ВР -** обнаружение резких (относительно кратковременных) изменений значений данных относительно некоторых стандартных, например предшествующих значений ряда. Такие значения могут иметь различную интерпретацию в том числе новизна в данных, выбросы, точки перегиба или другие.

**Корневыми причинами** аномалий могут быть, например, неисправности, мошенничество, появлене новых факторов или другие нерегулярные события.

Аномалии могут быть как

* **Локальными (**не превышают общий диапазон значений - **Контекстные**.)
* **Глобальными**
* **Точечными** .
* **Коллективными (**продолжительными**)**.

По характеру могут быть:

* **точечный скачок**
* **изменение дисперсии**
* **сдвиг уровня**
* изменение огибающей/сезонности/друой составляющей
* **кратковременное отсутствие паттернов или изменение**  поведения

**Методы обнаружения аномалий:**

* **контролируемые методы обнаружения аномалий** (класс нормы и аномалий известны)
* **полу-контролируемые методы обнаружения аномалий (**класс нормы известен**)**
* **неконтролируемые методы обнаружения аномалий (**по внешнему критерию**)**

**Глубокие нейронные сети** используются в случаях сложных ВР с не формулируемыми зависимостями при достаточно больших объемах данных. Часто это многопеременны ВР, где также сложно сформулировать связь компонент. При этом подходе:

* + - Нет необходимости в статистической гипотезе или конкретной форме модели.
    - Может аппроксимировать любую функцию с пропущенными данными, аномалиями и другими нерегулярными шаблонами.
    - Позволяют работать с огромными многомерными рядами данных со сложной взаимосвязью поведения между данными.
    - Автоматически извлекают и обрабатывают сложные признаки и отношения между ними.
    - Для сложных и многомерных данных в большом количестве модели, управляемые данными, могут обеспечить лучшую производительность.
  + **НО**
    - Требуется тщательная настройка гиперпараметров.
    - Часто требуется ансамбль сетей для получения высокой точности.
    - Тяжело перетренировать.
    - Трудно достичь сопоставимой точности с подходом на основе модели для относительно простых рядов.
  + Выбор конкретных методов зависит от поставленной задачи.

**Рекуррентные нейронные сети** это тип сетей, основанный на переиспользовании одних и тех же весовых параметров, но изменяющегося вектора состояний для последовательных участков данных.

Классический подход использования рекуррентных сетей (**vanilla RNN**) страдает от проблема взрыв веса (или градиента) и вымывание градиента в силу переиспользования параметров, особенно при длинном контексте. Также следствием высокой вероятности переобучения является не возможность сделать рекуррентные сети слишком глубокими. Для решения этой проблемы используют:

* усечение обучение (усеченное обратное распространение).
* двунаправленное обучение (если возможно).
* использование продвинутых архитектур рекуррентных сетей (LSTM, GRU и др.).
* регуляризация обучения (батч-норма, L1, L2,ограничение градиентные (или его нормы), функции активации с самонормализацией и т. д.).

Особенностью **LSTM** ячейки является раздельный учет скрытых состояний

* кратковременного контекста (например, так мы можем учесть сезонность),
* и долговременного контекста (так мы сможем учесть тренд, Долговременный контекст – это контекст с долгим затуханием).

Идея **GRU** ячейки объединить оба скрытых состояния вместе и передавать их одним параметром. За счет этого общее число параметров будет меньше.

**Сверточные сети** представляют аналог двух-мерных сверточных сетей для 1d случая. Популярны как простые архитектуры, так и использование таких приемов, как расширенная свертка и внимание.

# Заключение:

Модуль «Использование методов машинного обучения в анализе временных рядов» раскрывает особенности использования методов машинного обучения в анализе временных рядов. В том числе показаны основные особенности данных подходов в отношении этой модальности данных. Напомним, что эти особенности связаны с казуальностью и заданным порядком значений ВР, имеющим четкую интерпретацию. Именно это свойство и дает новую информацию, использование которой позволяет ввести те техники, примеры которых показаны в модуле.

Рассмотренные в модуле примеры задач в настоящее время имеют высокую степень важности, а методы для их решения актуальны и составляют т.н. State-of-the-art (то есть современный уровень решения). Однако, стоит понимать, что область знаний развивается достаточно быстро. Поэтому специализируясь на данной области нужно всегда поддерживать уровень знаний актуальным.