Санкт-Петербургский государственный университет

Гришутин Виталий Юрьевич

Выпускная квалификационная работа

Прогнозирование временных рядов при небольших объемах исходных данных

Уровень образования: бакалавриат
Направление 02.03.02 “Фундаментальная информатика и информационные технологии”
Основная образовательная программа “Программирование и информационные технологии”
Профиль “Автоматизация научных исследований”

Научный руководитель:
Кандидат технических наук,
доцент,
Гришкин В.М.

Рецензент:
Доктор технических наук,
Профессор,
Буре В.М.

Санкт-Петербург
2020

Содержание

Введение

Прогнозирование посредством представления временными рядами рассматриваемых процессов, изменяющихся характеристик или других данных, зависящих от времени, используется во многих областях: это и бизнес, экономика, инженерия, физика, химия, метеорология, социология и др. В частности, наблюдения в каком-либо химическом процессе, цены акций конкретной компании на бирже, прогноз погоды, количество дорожно-транспортных происшествий, данные о количестве проданных товаров и полученной прибыли и многое другое можно охарактеризовать временными рядами. Тем самым, прогнозирование этих рядов является очень важной задачей, т.к. она позволяет нам получить оценку будущих наблюдений заранее, что несет выгоду разного характера.

Временные ряды обычно моделируются посредством *счетного стохастического процесса* $Y(t),\ t \in N$, то есть счетной совокупности случайных величин, при этом, этот процесс зависит от времени. В условиях прогнозирования мы находимся в момент времени *t*, и мы заинтересованы в оценке $Y(t_{\text{нач}} + t),\ t_{\text{нач}}, t \in N$, используя только информацию, доступную в полуинтервале времени $[t_{\text{нач}}; t_{\text{нач}} + t)$. Длиной временного ряда чаще всего обозначают количество наблюдений или время от начала до конца процесса. Подробно об этом в учебнике Броквелла и Девиса [1].

Точность при прогнозировании будущих наблюдений зависит от количества известных нам наблюдений, то есть от длины полуинтервала $[t_{\text{нач}}; t_{\text{нач}} + t)$. . Чаще всего, компании, имеющие данные о продажах своего товара за несколько лет, могут спрогнозировать прибыль в следующем месяце намного лучше, чем только что открывшиеся компании, проработавшие пару месяцев. Подобную закономерность можно увидеть и в других сферах жизни. Именно в этом кроется проблема небольших объемов исходных данных- не в количестве рядов, а в их длине.

Для прогнозирования временных рядов используются десятки методов: от наивных и регрессионных моделей до нейронных сетей. Все эти модели при большом количестве наблюдений дают лучший результат (за исключением наивных моделей), и есть очень много материалов об их эффективности на рядах большой длины, что позволяет выбрать, какую модель лучше использовать. Но что делать, если количество наблюдений крайне мало, и какую модель регрессии выбрать для прогнозирования будущих значений? В этой работе будут рассмотрены данные вопросы.

Постановка задачи

Рассматривая вышеуказанные вопросы, требуется реализовать несколько основных моделей регрессии для прогнозирования временных рядов, использовать их на большом количестве рядов небольшой длины, эмпирически подобрать лучшие параметры для каждой модели, провести сравнительный анализ моделей и выбрать те, которые будут иметь лучшую точность на этих рядах и попутно рассмотреть эти модели с точки зрения малого объема исходных данных. Как результат, утвердить лучшие модели как оптимальные для прогнозирования временных рядов небольшой длины.

Обзор литературы

Многие работы, связанные с прогнозированием временных рядов и использующие разнообразные регрессионные модели для оценки будущих наблюдений, рассматривают временные ряды большой длины, обычно превышающей 50, а в большинстве своем, достигающей нескольких сотен. Такие ряды, к примеру, рассматривает Хиндман [2] и Бурба [3] в своих работах. Количество таких материалов превалирует над обзорами рядов небольшой длины. Работы же, учитывающие проблему небольших исходных данных, либо рассматривают такие временные ряды в рамках одной модели регрессии (тот же Хиндман [2]), либо используют совсем другие подходы, не имеющие отношение к сравнительному анализу. К примеру, в учебнике Урла [4], используется экономический подход, с вычислением трендов и сезонности на очень маленьком стеке примеров. Таким образом, материалы с рядами небольшой длины обычно имеют 2 недостатка: нет сравнения хотя бы нескольких основных моделей регрессии и нет проверки их работы на большом количестве временных рядов. Выполнив нашу задачу, эти недостатки будут убраны.

Что касается выбора самих моделей прогнозирования, на основе обзора литературы было принято решение использовать несколько основных моделей регрессии, которые чаще всего используются для прогнозирования временных рядов и подвергаются сравнительному анализу во многих работах, публикация Бурбы [3] и учебник Бокса [5] – одни из примеров. Ниже приведены данные модели:

- “Наивная” модель, которая приравнивает все будущие наблюдение последнему известному, рассматривая ее как опорную отметку, сравнивая последующие модели с ней и объявляя модели, имеющие точность меньше, чем она, неоптимальными для прогнозирования рядов небольшой длины.

- Авторегрессионная модель, которая указывает, что будущие наблюдения линейно зависят от предыдущих.
- Модель скользящего среднего, где значения будущих наблюдений вычисляются посредством линейной регрессии “белых шумов”.
- Модель авторегрессии — скользящего среднего, являющейся комбинацией предыдущих двух моделей.
- Интегрированная модель авторегрессии — скользящего среднего, которая является модификацией предыдущей модели, где все значения наблюдений заменяются на разность их значений с предыдущим наблюдением.
- Модель векторной авторегрессии, обобщающая авторегрессионную модель, используя временной ряд несколько раз для вычисления будущих значений.
- Модель экспоненциального сглаживания, вычисляющая значения будущих наблюдений при помощи функции “экспоненциального окна”
- Модель авторегрессионной условной гетероскедастичности, для вычислений использующая условную дисперсию.
- Модель обобщенной авторегрессионной условной гетероскедастичности, которая является предыдущей моделью, с предположением о том, что имеется зависимость условной дисперсии от своих предыдущих значений.

В дополнение к этому, для прогнозирования будут рассмотрены и нейронные сети разных видов, а именно:

- Обычная нейронная сеть на основе многослойного перцептрона, где на каждом слое нейронная сеть обучается, используя функции активации и на выходе получая некоторые веса, по которым находятся значения будущих наблюдений у временных рядов в тестовой выборке.
- Рекуррентная нейронная сеть, в которой соединения между узлами образуют ориентированный граф, который можно представить, как

последовательность. В частности, сеть, использующая долгую краткосрочную память (LSTM).

- Сверточная нейронная сеть, которая, хоть и используется чаще всего для работы с изображениями, также рассматривается в прогнозировании. В ней ряды создают двумерную свертку, по которой с помощью скользящего ядра вычисляются веса.

Нейронные сети этих трех видов в качестве моделей рассматривал Браунли [6].

Глава 1. Модели прогнозирования временных рядов

В данной главе будут описаны модели, которые были выбраны в рамках обзора литературы. Самое главное в чем нужно разобраться в рамках нашей работы – от чего зависит точность прогноза у каждой модели, в частности, зависит ли она от длины временного ряда. Перед тем, как их рассматривать, будем полагать, что $t_{\text{нач}} = 0$, то есть будем прогнозировать временные ряды используя все значения каждого известного наблюдения в них, т.к. небольшая длина рядов не позволяет не обращать внимание даже на первые наблюдения. Следовательно, мы будем заинтересованы в оценке $Y(t)$ по значениям $Y(0), Y(1), \dots, Y(t-1)$.

1.1 Модель Naïve.

Наивный подход заключается в том, что все прогнозы равны последнему известному наблюдению:

$$Y(x \geq t) = Y(t-1)$$ [7]

Но зачем нужна такая простая модель? Ответ заключается в том, что наивная модель намного легче всех остальных моделей и требует меньших ресурсов, таких как время на ее реализацию и время. При этом она может превзойти некоторые другие модели по точности прогноза. Сравнивая другие модели с ней, можно исключать модели из рассмотрения, которые имеют точность меньше чем наивная, т.к. при больших затратах, не имеет смысла использовать модели, которые хуже наивной. [8]

В случае небольших исходных данных наивная модель имеет огромное преимущество над всеми остальными моделями регрессии, т.к. значения прогнозируемых наблюдений зависят лишь от последнего наблюдения и, таким образом, независимы от длины временного ряда.

Недостатком модели является то, что она совсем не учитывает закономерности во временных рядах, и с ее помощью нельзя добиться большой точности прогноза.

1.2 Авторегрессионная модель (AR)

В этой модели мы вычисляем значения прогнозируемого наблюдения последующей формуле:

$$Y(t) = \Sigma_{\mathrm{i}=1}^{\mathrm{p}} \beta_i Y(t-i) + \varepsilon_t,$$

где β_i,$\Sigma_{\mathrm{i}=1}^{\mathrm{p}} \beta_i = 1$ – коэффициенты регрессии, ε_t – один из элементов белого шума (подробнее о нем в следующей модели). [9] [10]

Параметр p в данном случае показывает порядок авторегрессии AR(p). Значение прогнозируемого наблюдения зависит от p последних наблюдений. Чаще всего, чем больше значение p, тем больше точность прогноза, но в нашем случае мы не можем позволить большого значения p, ведь длина временного ряда будет небольшой. Увеличивая порядок авторегрессии, мы будем убирать из рассмотрения ряды, имеющие длину меньше этого порядка. Но это свойство модели является не только недостатком, но и достоинством. Ведь при небольшом значении порядка, можно прогнозировать временные ряды в условиях небольших исходных данных.

1.3 Модель скользящего среднего (MA)

В данной модели мы вычисляем значение будущих наблюдений по взвешенной сумме белого шума:

$$Y(t) = \varepsilon_t + \Sigma_{\mathrm{i}=1}^{q} \theta_{\mathrm{i}} \varepsilon_{t-i},$$

где θ_{i}, $\Sigma_{\mathrm{i}=1}^{q} \theta_{\mathrm{i}} = 1$ – коэффициенты регрессии, $\{\varepsilon_t\}$ – последовательность, называемая белым шумом. [10]

Параметр q в данном случае показывает порядок авторегрессии MA(q). В модели MA(q) последовательность $\{\varepsilon_t\}$ является нормальным разбиением вероятностей порядка N(0, q). Значение прогнозируемого наблюдения зависит от всех известных наблюдений, к примеру, в модели порядка 1 оно будет равно:

$$Y(t) = \varepsilon_t - \Sigma_{\mathrm{i}=1}^{t-1} \theta_1^i Y(t-i)$$

Это является существенным недостатком, ведь при прогнозировании рядов небольшой длины, эта модель теряет свою эффективность.

1.4 Модель авторегрессии — скользящего среднего (ARMA)

Модель является комбинацией предыдущих двух моделей:

$$Y(t) = \Sigma_{\mathrm{i}=1}^{\mathrm{p}} \beta_i Y(t-i) + \varepsilon_t + \Sigma_{\mathrm{i}=1}^{q} \theta_{\mathrm{i}} \varepsilon_{t-i},$$

где $\beta_i, \Sigma_{\mathrm{i}=1}^{\mathrm{p}} \beta_i = 1$ и $\theta_{\mathrm{i}}, \Sigma_{\mathrm{i}=1}^{q} \theta_{\mathrm{i}} = 1$– коэффициенты регрессии, , $\{\varepsilon_t\}$ – последовательность, называемая белым шумом. [10]

Модель ARMA(p, q) имеет те же достоинства и недостатки, что и предыдущие модели, но при этом у нее есть одна особенность: подобрав каким-либо способом параметры p и q, можно добиться точности прогноза лучше, чем AR и MA по отдельности.

1.5 Интегрированная модель авторегрессии — скользящего среднего (ARIMA)

Данная модель является модификацией предыдущей модели, добавляя в нее возможность продифференцировать члены временного ряда несколько раз перед самим прогнозированием. В модель ARIMA(p,d,q) добавляется параметр d- количество поочередных дифференцирований. Значения наблюдений дифференцируются по следующим формулам:

$$d = 0: Y(i) = Y(i)\,, i \in [1, t-1],$$

$$d = 1: Y(i) = Y(i) - Y(i-1)\,, i \in [2, t-1], Y(1) - \text{ не меняется},$$

$$d = 2: Y(i) = \big(Y(i) - Y(i-1)\big) - \big(Y(i-1) - Y(i-2)\big), i \in [3, t-1],$$

$$Y(1) \text{и } Y(2) - \text{не меняются, и т. д.}$$

Для прогнозирования наблюдений используется та же формула, что и у ARMA:

$$Y(t) = \Sigma_{\mathrm{i}=1}^{\mathrm{p}} \beta_i Y(t-i) + \varepsilon_t + \Sigma_{\mathrm{i}=1}^{q} \theta_{\mathrm{i}} \varepsilon_{t-i}$$

Модель имеет точность прогноза больше, либо равную точности ARMA, при этом сохраняя те же достоинства и недостатки. [11]

1.6 Модель векторной авторегрессии (VAR)

Данная модель представляет из себя реализацию модели авторегрессии для временных рядов размерности больше $n > 1$[12]:

$$Y_1(t) = \Sigma_{i=1}^{p} \beta_i Y_1(t-i) + \varepsilon_t,$$

$$Y_2(t) = \Sigma_{i=1}^{p} \beta_i Y_2(t-i) + \varepsilon_t,$$

$$\dots$$

$$Y_n(t) = \Sigma_{i=1}^{p} \beta_i Y_n(t-i) + \varepsilon_t,$$

где $Y_i(t)$ – значение наблюдения под номером t в i-том ряду.

Но каким образом такую модель можно использовать на рядах размерности 1? Как вариант: преобразовать все значения наблюдений в массивы, состоящие из n соответствующих значений, к примеру 2. Данная модель VAR(p) будет иметь другие коэффициенты регрессии, в отличие от AR и результат будет другим. Как и модель авторегрессии обладает теми же достоинствами и недостатками в условиях небольших исходных данных.

1.7 Модель экспоненциального сглаживания (SES)

При прогнозировании конкретного ряда модель создает сглаживаемый ряд $L(t)$, используя следующую рекурсию:

$$L(t-1) = \alpha Y(t-1) + (1-\alpha) L(t-2),$$

$$L(1) = Y(1),$$

где α – коэффициент сглаживания.[13]

Найдя значение $L(t-1)$, мы полагаем $Y(t) = L(t-1)$.

Значение прогнозируемого наблюдения зависит от всех предыдущих наблюдений. В условиях небольших исходных данных- это является минусом. При создании данной модели также требуется подобрать коэффициент сглаживания.

1.8 Модель авторегрессионной условной гетероскедастичности (ARCH)

Данная модель использует формулу условной дисперсии как ниже приведенную рекурсию:

$$Y(t) = \sigma_t \varepsilon_t,$$

$$\sigma_t = \sqrt{a_0 + \Sigma_{\mathrm{i}=1}^{\mathrm{p}} a_i Y^2(t-i)},$$

где σ_t – условная дисперсия p последних членов временного ряда, $a_i > 0, \Sigma_{\mathrm{i}=1}^{\mathrm{p}} a_i = 1$ - коэффициенты регрессии.[14]

Рассматривая эту рекурсию видно, что значение прогноза зависит от p последних наблюдений. Следовательно, при небольшой длине временных рядов модель ARCH(p) имеет такие же достоинства и недостатки, как и модель AR. Стоит также отметить, что модель чаще всего используют для финансовых нестационарных временных рядов, с большой длиной, трендом и сезонностью, т.к. у стационарных рядов дисперсия крайне мала. Для нашего случая она может не подойти.

1.9 Модель обобщенной авторегрессионной условной гетероскедастичности (GARCH)

Эта модель основана на предыдущей, но здесь вводится предположение о том, что условная дисперсия зависима от своих предыдущих значений. Формула рекурсии также меняется:

$$Y(t) = \sigma_t \varepsilon_t,$$

$$\sigma_t = \sqrt{a_0 + \Sigma_{\mathrm{i}=1}^{\mathrm{p}} a_i Y^2(t-i) + \Sigma_{\mathrm{j}=1}^{q} \beta_j \sigma_{t-j}^{j}},$$

где $\beta_j > 0, \Sigma_{\mathrm{j}=1}^{q} \beta_j = 1$ – коэффициенты регрессии.[14]

Параметр q показывает количество предыдущих значений от которых зависит условная дисперсия. GARCH(p,q) имеет те же плюсы и минусы, что и

ARCH в условиях небольших исходных данных, но в большинстве случаев прогнозирует значения немного лучше.

1.10 Нейронная сеть на основе многослойного перцептрона (MLP)

Вкратце разберемся, как работает данная нейронная сеть. Для начала мы разобьем наши временные ряды на 2 выборки: тренировочную и тестовую. Во входном слое сеть будет принимать значения всех известных наблюдений одного ряда из тренировочной выборки, далее в скрытых слоях будут задаваться веса, потом используя функции активации выходной значение нейрона либо учитывается, либо нет. Дойдя до выходного слоя вычисляются значения в них. Структура сети показана на рис 1.1. С помощью обратного распространения ошибки веса будут принимать другие значения, чтобы ошибка между полученным и фактическим ответами минимизировалась. Данный цикл называется эпохой.

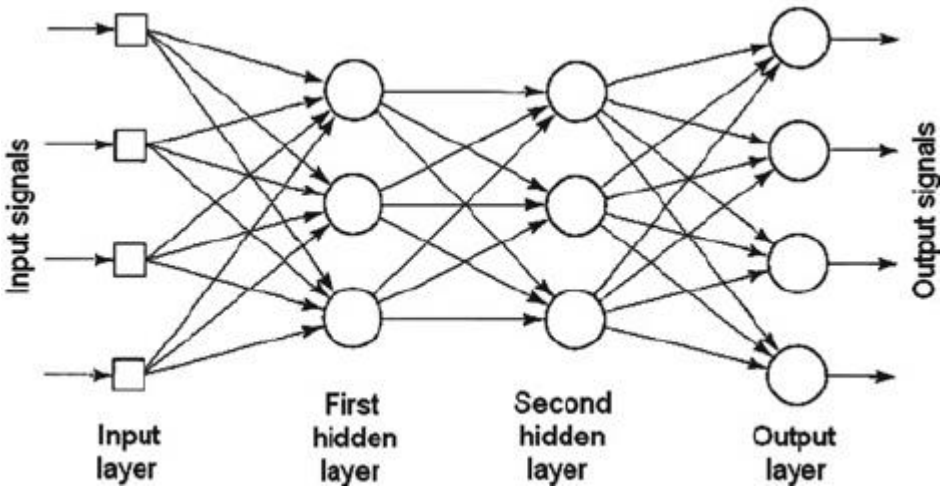


Рис. 1.1 Схема многослойного перцептрона

Далее с помощью этих же весов прогнозируются ряды в тестовой выборке, и уже по ним вычисляется точность прогноза данной сети [15]. В подробности этих механизмов вдаваться не будем, есть множество материалов по ним, нас лишь интересует от чего зависит точность прогноза.

Точность прогноза зависит не только от длин временных рядов, но и от количества эпох, выбора ошибки, количества нейронов в слоях и количества

самих слоев, выбора функций активации и разбиения на выборки, ведь чем больше тренировочная выборка, тем лучше сеть прогнозирует на тестовой. Следовательно, нейронная сеть в теории подойдет для прогнозирования временных рядов при небольших исходных данных, ведь она зависит от многих других параметров. Недостатком нейронных сетей является то, что для них обязательно наличие большого количества рядов в тренировочной выборке, и, таким образом, она не сможет прогнозировать все временные ряды, в отличие от предыдущих моделей регрессии.

1.11 Рекуррентная нейронная сеть на основе долгой кратковременной памяти (LSTM)

Рекуррентные нейронные сети отличаются от предыдущих сетей обратной связью, то есть нейроны могут передавать значения не только нейронам в следующем слое, но и самим себе. Такая модификация позволяет сетям лучше работать с последовательностями, а временные ряды ими и являются. LSTM же является рекуррентной нейронной сетью, где на каждом шагу обучения учитываются не только недавние зависимости, но и те, что были найдены давно, это позволяет устранить проблему долговременных зависимостей, которой обладают другие рекуррентные сети. Структура сети LSTM показана на рис. 1.2. [16]

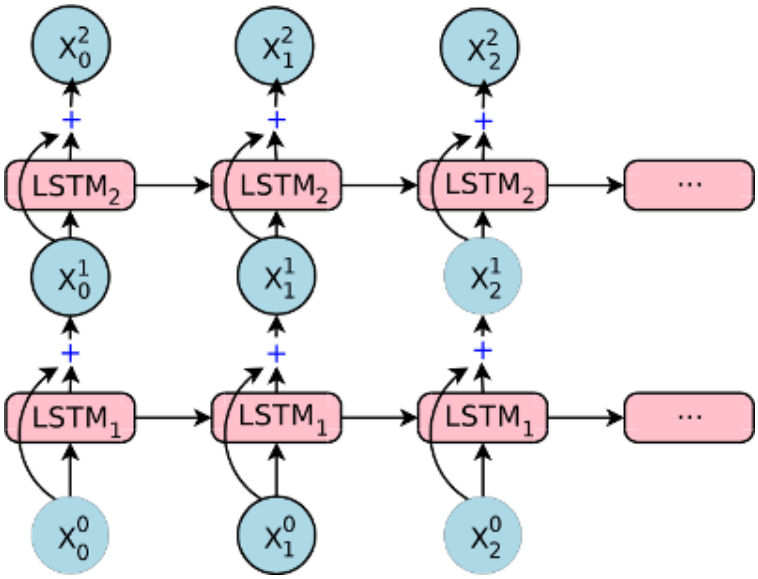


Рис 1.2 Схема LSTM

Точность прогноза LSTM зависит от тех же параметров, что и MLP. Основной недостаток нейронных сетей, невозможность прогнозирования всех рядов, все также присутствует. Но помимо этого, благодаря своей особенности, LSTM может прогнозировать временные ряды неопределенной длины. Правда, в случае небольших исходных данных это преимущество сводится на нет.

1.12 Сверточная нейронная сеть (CNN)

Сверточные нейронные сети, хоть и предназначавшиеся для работы с изображениями, могут быть использованы для прогнозирования временных рядов. Для их реализации необходимо будет сделать из рядов в тренировочной выборке двумерную свертку, то есть матрицу, где на каждой строке будут значения наблюдений одного ряда. Далее ядра, являющиеся матрицами весов, проносятся по всей матрице и создают новые матрицы взвешенных сумм весов. Структура приведена на рис. 1.3-1.5. С помощью полученных матриц прогнозируются значения будущих наблюдений у рядов в тестовой выборке. [17]

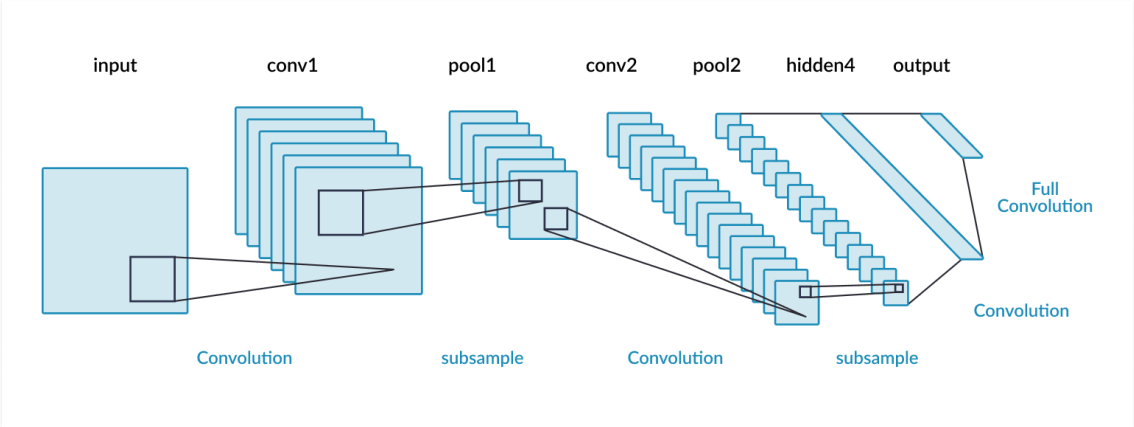


Рис. 1.3 Схема CNN

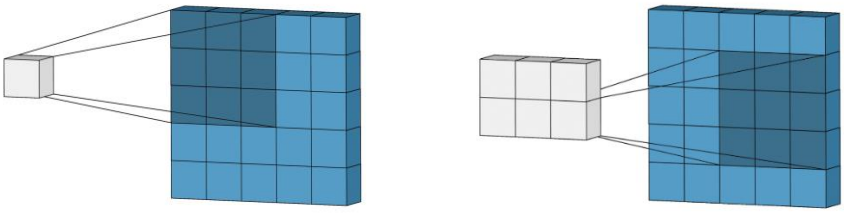


Рис 1.4 и 1.5 Прохождение ядра по матрице наблюдений

Точность прогноза зависит от длин временных рядов, количества ядер и их размеров, количества эпох, выбора функций активации и ошибки. По той же логике, как и прошлые нейронные сети, подойдет для прогнозирования временных рядов небольшой длины.

Глава 2. Реализация выбранных моделей

В данной главе будут рассмотрены временные ряды, на которых модели будут работать, предобработка исходных данных, выбор программных инструментов с обоснованием и структура программной реализации.

2.1 Обзор и предобработка исходных данных

Вкратце рассмотрим исходные данные. Для проверки моделей были предоставлены 1146 временных рядов от компании Huawei в виде csv-таблиц, в одной таблице были значения спроса за несколько месяцев для одного товара, находящиеся в промежутке [0;1] (в 3 столбце на изображении ниже), количество этих значений не превышает 47 в любой таблице (рис 2.1):

	A	B	C
1	period_id	item	qty
2	1213	item1213	0.2447
3	1214	item1213	0.198
4	1215	item1213	0.5205
5	1216	item1213	0.1627
6	1217	item1213	0.2037
7	1218	item1213	0.0764
8	1219	item1213	0.1301
9	1220	item1213	0.0707
10	1221	item1213	0.2122

Рис 2.1 Пример таблицы с временным рядом

Для прогнозирования будут отводиться значения последних 4 наблюдений у каждого временного ряда, по полученным и фактическим значениям будет вычисляться точность по следующей формуле:

$$\text{точность} = \frac{min(\text{фактическое значение}, \text{прогноз})}{max(\text{фактическое значение}, \text{прогноз})},$$

$$\text{если фактическое значение} = \text{прогноз} = 0, \text{то точность} = 100\%$$

После этого для каждой модели будет вычисляться средняя точность по всем прогнозам. Полученное значение будет показывать эффективность модели.

Для правильной реализации моделей регрессии перед этим исходные данные следует подвергнуть предобработке. Для начала были убраны из

рассмотрения 9 таблиц, длина рядов которых была меньше 5, т.к. модели будут предсказывать значения последних 4 наблюдений, и для этого им нужны как минимум 5 значений.

Далее нужно проверить временные ряды на стационарность. Если некоторые ряды окажутся нестационарными (т.е. в них есть некоторая сезонность или тренд), на них можно будет использовать сезонные аналоги моделей регрессии, к примеру, SNaïve или SARIMA. Также на нестационарных рядах модели ARCH и GARCH вычисляют значение прогнозируемого наблюдения намного лучше. Сезонность во временных рядах – это некоторая периодичность в значениях, а тренд – это тенденция изменения значений наблюдений во времени, то есть возрастание или убывание.

Для проверки на стационарность применяется KPSS-тест. Проверяя каждый ряд, он выдвигает гипотезу о том, что ряд является стационарным. Далее тест вычисляет значение p_value, оно зависит от сезонности и тренда, причем, если сезонность и тренд ярко выражены, то p_value будет меньше. Вводится уровень значимости $\alpha = 0.05$ и если $p\_value < \alpha$, то гипотеза о стационарности отклоняется. [18] Проверив все 1146 таблиц, все временные ряды оказались стационарными (рис 2.2):

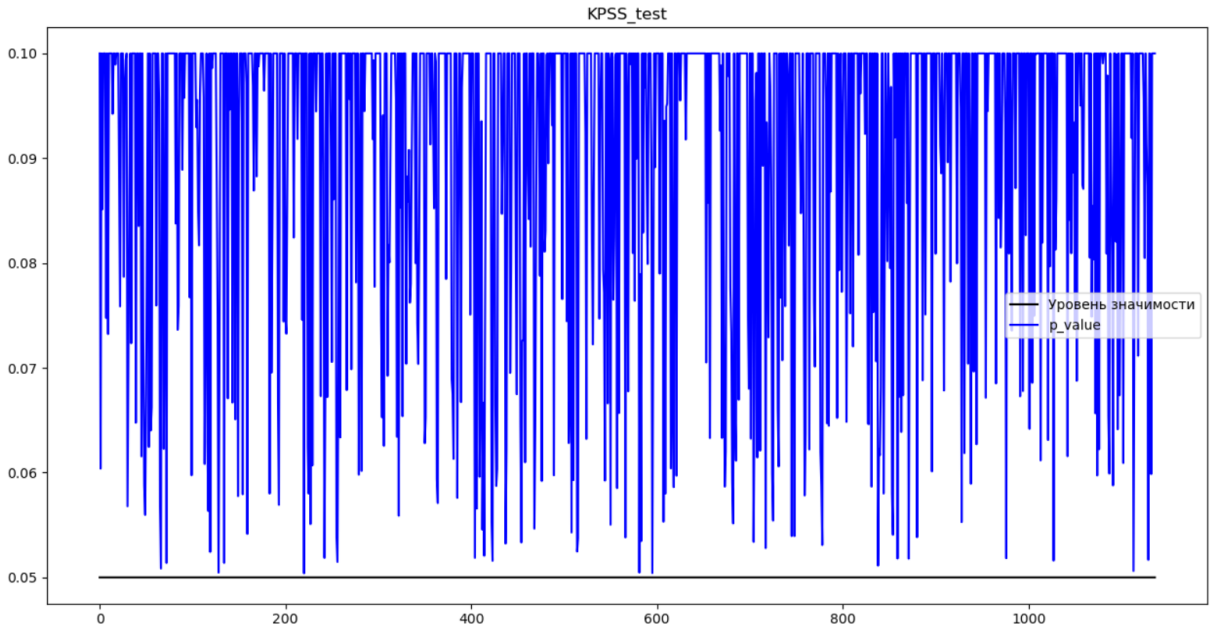


Рис 2.2 Результаты KPSS-Теста

Из этого можно сделать вывод о том, что использование моделей, учитывающие тренд и сезонность на данных рядах окажется неэффективным. Это будет также показано на результатах моделей ARCH и GARCH в следующей главе.

2.3 Выбор программных инструментов

Для реализации всех моделей был выбран язык программирования Python версии 3.6, т.к. для него написано множество библиотек использующиеся для машинного обучения, в частности те, которые будут приведены ниже.

Чтобы считывать исходные данные из таблиц, применяется библиотека “csv”, ввиду их расширения. Для создания графиков используется библиотека “matplotlib”, она является самым простым и очень популярным для этого инструментом. Для создания моделей регрессии AR, MA, ARMA, ARIMA, VAR и SES была выбрана библиотека “statsmodels”. В ней есть все инструменты для создания этих моделей, их прогноза, а также благодаря этой библиотеке можно с легкостью менять параметры модели. Библиотека “arch” позволяет смоделировать модели ARCH и GARCH. Ее возможности идентичны библиотеке “statsmodels” и она легка в понимании. Для создания нейронных сетей MLP, LSTM и CNN применяется библиотека “keras”. Она предоставляет пользователю все инструменты для создания нейронных сетей, в частности методы оптимизации, функции активации, легкое создание слоев нейронов, обучение сети и смену параметров обучения. Чтобы разобраться в этих библиотеках, была прочитана документация. Для разбиения рядов на тренировочную и тестовую выборку понадобилась библиотека “numpy”.

2.4 Структура программной реализации

Чтобы реализовать все модели, для начала нужно прочитать временные ряды с таблиц, при этом исключая ряды, длина которых меньше 5, т.к. модели прогнозируют 4 последних значения). Используя библиотеку csv данные

считывались тремя способами: по одному ряду для всех моделей кроме нейронных сетей и VAR, по двум одинаковым рядам для VAR и двумерным массивом для нейронных сетей.

В наивной модели для каждого ряда прогноз был равен 5-му ряду с конца, то есть значению последнему известному наблюдению спроса товара. После вычислялась точность для каждого прогноза и в конце средняя точность.

Далее рассмотрим все модели кроме нейронных сетей. Для каждого ряда создавался цикл в 4 итерации для прогнозирования 4 последних значений. С помощью библиотек “statsmodels” и “arch” в цикле создавались модели регрессии, не учитывая последние 4 значения. Для установки модели и прогнозирования значения применялись методы fit и predict. Параметры указывались при создании модели или в этих методах, в зависимости от модели. По полученному значению на каждой итерации вычислялась точность. Если по некоторым причинам прогноз был невозможен, к примеру, из-за разнообразных ошибок в прогнозировании значения, точность прогноза ставилась равной нулю при помощи конструкции try-except. Самый частый пример ошибки в прогнозировании- высокое значение параметра p в моделях AR, ARMA, ARIMA и VAR, где прогноз строился по p значениям последних известных наблюдений спроса товара, при этом количество этих наблюдений было меньше p. Также ввиду того, что значения наблюдений в рядах лежат в промежутке [0;1], в каждую модель была добавлена следующая модификация: если значение прогноза – отрицательное число, то оно приравнивается 0, и аналогично 1, если больше 1. После прогнозирования всех рядов находилась средняя точность.

При прогнозировании с помощью нейронных сетей множество рядов разбивалась на 2 выборки: тренировочную и тестовую. Рассматривались 2 разбиения: 8:2 и 9:1. Для работы LSTM и CNN выборки также преобразовывались в трехмерные массивы, используя метод reshape. С помощью библиотеки “keras” создавались слои для нейронных сетей. Для MLP

несколько раз использовался метод Dense, который создавал один слой нейронов. Каждому слою выбиралась функция активации и количество нейронов. Для LSTM- метод LSTM. Он создавал слой с обратной связью и долгой кратковременной памятью, где также можно было выбрать количество нейронов и функцию активации. Для моделирования CNN применялся метод Conv1D, где задавались количество ядер, их размеры и функции активации. Далее каждая нейронная сеть компилировалась, где выбиралась функция оптимизации (для всех был выбран Adam) и ошибка. Обучение модели реализовывалось с помощью метода fit, в котором указывалось количество эпох. В конце вычислялась средняя точность модели.

Программный код для каждой модели с комментариями на языке Python находится в приложении, в конце данной работы.

Глава 3 Экспериментальное исследование моделей

В данной главе будут показаны точности работы моделей при разных параметрах и среди них будет выбрана модель с наибольшей точностью, которую мы посчитаем оптимальной для прогнозирования временных рядов в условиях небольших исходных данных. Параметры будут подбираться эмпирически, т.к. в моделях регрессии количество параметров мало, а в нейронных сетях автоматизировать этот процесс крайне трудно. Исключением будет модель SES, ведь ее параметр – вещественное число Средняя точность будет округлена до сотых процента.

3.1 Исследование моделей с учетом их параметров

Naïve: В наивной модели нет параметров, которые можно изменить и получить другие значения при прогнозе. Средняя точность прогноза после прогона по всем временным рядам равна 54.15%.

AR: Авторегрессионная модель AR(p) позволяет поменять один параметр $p \geq 1$ для изменения прогнозов. Ниже приведены значение средней точности при разном значении параметра p:

$p = 1$: 55.13%, $p = 2$: 54.91%, $p = 3$: 54.68%, $p = 4$: 53.22%.

Увеличивая значение параметра дальше, точность продолжает падать. Лучший результат модель показала при $p = 1$, это лучше, чем результат наивной модели, значит, следующие модели будут сравниваться с AR.

MA: Как и в предыдущей модели, в модели скользящего среднего MA(q) есть один параметр. Ниже приведены значение средней точности при разном значении параметра $q \geq 1$:

$q = 1$: 52.23%, $q = 2$: 51.82%, $q = 3$: 51.86%, $q = 4$: 50.32%.

Аналогично авторегрессионной модели, MA(q) имеет лучшее значение средней точности при $q = 1$, но точность не дотягивает до AR.

Для наглядности ниже показан график фактических и прогнозируемых значений наблюдений первых 3 моделей для 25 рядов (рис. 3.1):

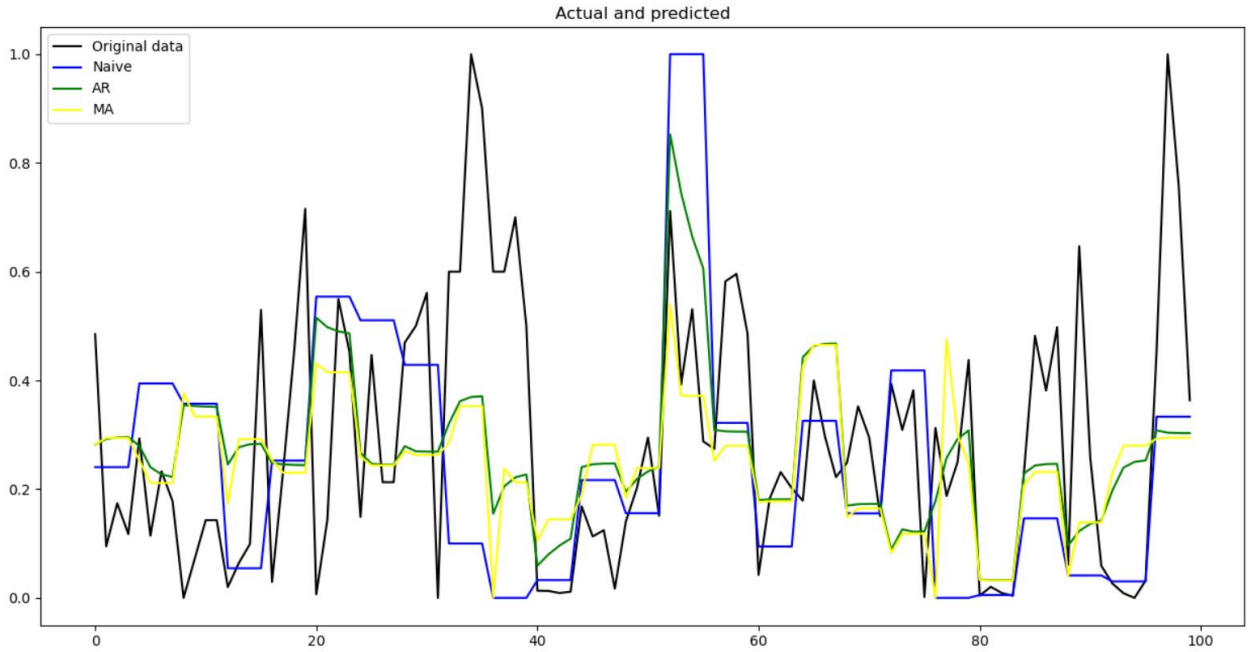


Рис 3.1 График прогнозов моделей Naïve, AR и MA для 25 рядов.

ARMA: Комбинация предыдущих двух моделей зависит от двух параметров $p \geq 1$ и $q \geq 1$. Если параметр $p = 0$, то данная модель станет моделью MA, если $q = 0$, то AR. Таким образом, параметры всегда должны быть положительными. Ниже приведены значение средней точности при разных значениях параметров модели ARMA(p,q):

	$q = 1$	$q = 2$	$q = 3$
$p = 1$	56,56%	55,46%	54,36%
$p = 2$	55,19%	55,35%	53,81%
$p = 3$	55,02%	53,58%	53,76%

Увеличение параметров за эти значения также ухудшает результат. Следовательно, лучше всего брать $p = 1$ и $q = 1$. Средняя точность равна 56.56%, это лучше, чем результат модели AR, следовательно, ARMA на данный момент является оптимальной для прогнозирования.

ARIMA: В этой модели добавляется новый параметр $d \geq 1$. В отличие от предыдущей модели, в этой параметры p и q уже могут быть равны 0 (но не

одновременно), т.к. она будет выдавать результат, отличный от AR и MA, из-за d поочередных дифференцирований. При подборе параметров выяснилось, что при $d \geq 2$ точность прогнозирования у модели резко падало. Ниже приведены значения точности ARIMA(p,d,q) при разных параметрах p и q:

	$q = 0$	$q = 1$	$q = 2$	$q = 3$
$p = 0$	-	58.14%	58.01%	58.31%
$p = 1$	55.87%	57.79%	56.65%	54.81%
$p = 2$	56.35%	57.3%	55.83%	54.18%
$p = 3$	56.33%	56.14%	55.05%	52.65%

Далее наблюдается такая же тенденция – при увеличении значений параметров p и q средняя точность прогноза уменьшается. Лучшая достигается при $p = 0,\ d = 1,\ q = 3$ – 58.31%. Результат лучше, чем у предыдущей модели, значит, ARIMA становится оптимальной моделью.

VAR: Векторная авторегрессия имеет параметр модели AR- $p \geq 1$. Ниже приведены результаты при разных значениях параметров модели VAR(p):
$p = 1$: 55.11%, $p = 2$: 55.02%, $p = 3$: 55.02%, $p = 4$: 53.68%.

Если увеличивать значение параметров и дальше, то точность будет падать. В итоге лучший результат модель показывает при $p = 1$. Но он хуже, чем у предыдущей модели, поэтому ARIMA остается оптимальной.

Для ARMA, ARIMA и VAR получен аналогичный график (рис 3.2):

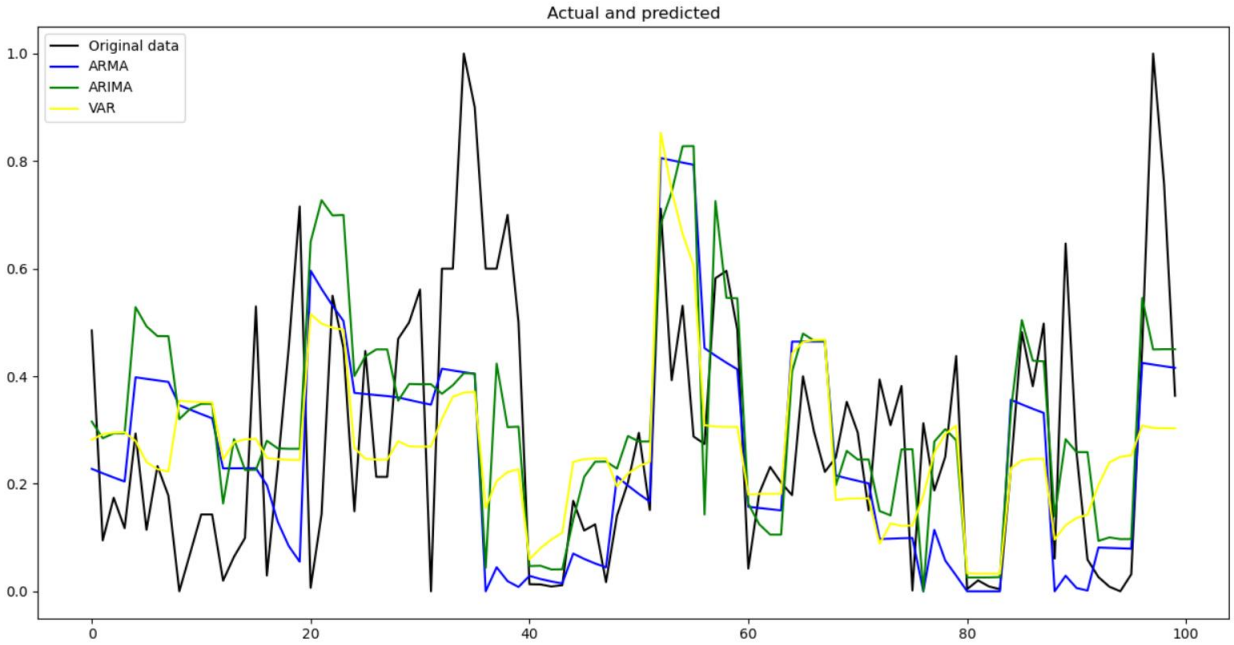
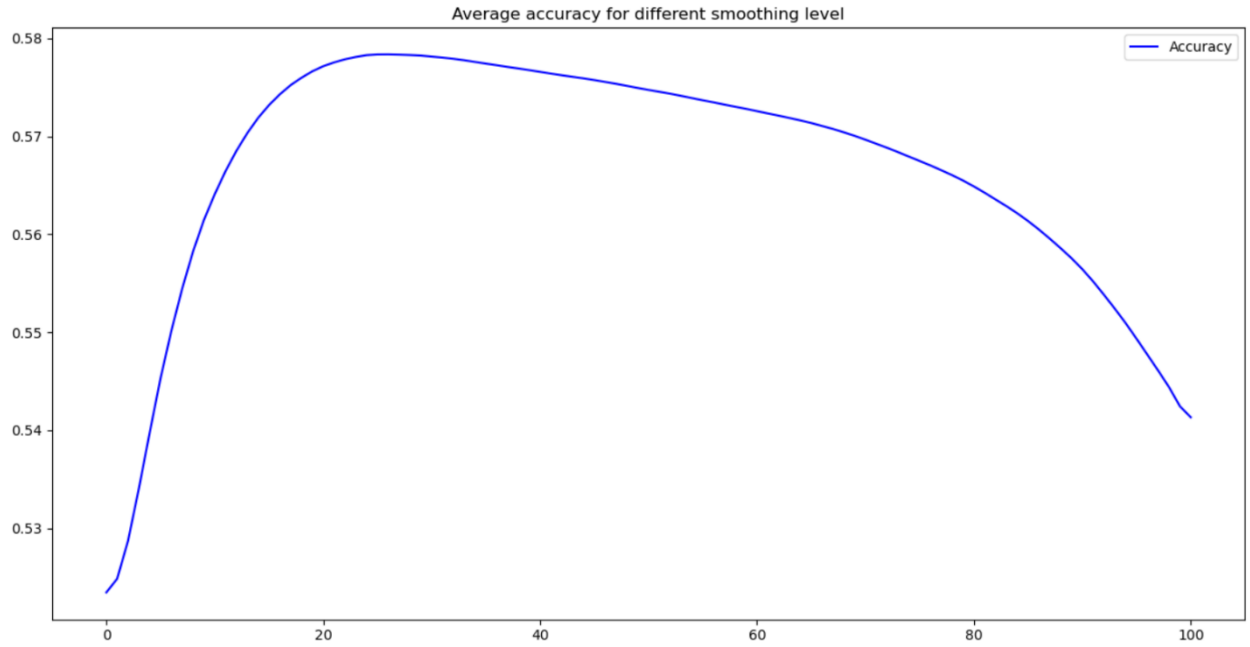


Рис 3.2 График прогнозов моделей ARMA, ARIMA и VAR для 25 рядов.

SES: Параметр данной модели – коэффициент сглаживания $\alpha \in [0; 1]$, который является вещественным числом. Для поиска значения коэффициента, при котором модель показывает лучшее значение средней точности был создан цикл, который проверяет работу данной программы на промежутке $\alpha \in [0; 1]$ изменяя значение коэффициента с шагом 0.01. Полученные значения точности приведены на графике:



Лучший результат: 57.84% при $\alpha = 0.26$. Результат хороший, но хуже, чем у ARIMA.

ARCH: Средняя точность прогноза ARCH(p) зависит от параметра p – количество последних членов ряда, по которым высчитывается значение условной дисперсии. Ниже приведены результаты при разных значениях параметра p:

$p = 1$: 34.42%, $p = 2$: 34.21%, $p = 3$: 34.13%, $p = 4$: 34.16%, $p = 5$: 34.09%.

Дальше значение средней точности прогноза уменьшается, аналогично предыдущим моделям. Как видно, особенность исходных данных, а именно стационарность рядов, не позволяет показать ARCH хороший результат, как и было предположено ранее.

GARCH: К этой модели добавляется еще один параметр q - количество предыдущих значений условной дисперсии от которых она зависит. Ниже приведены результаты работы GARCH(p,q) при разных значениях параметров:

	$p = 1$	$p = 2$	$p = 3$	$p = 4$	$p = 5$
$q = 1$	36.88%	37.77%	37.89%	38.26%	37.63%
$q = 2$	36.45%	36.88%	37.06%	37.44%	37.37%

Ситуация схожа с прошлой моделью. Модель GARCH не показала хороших результатов.

График прогнозов моделей SES, ARCH, GARCH для 25 временных рядов (рис 3.3):

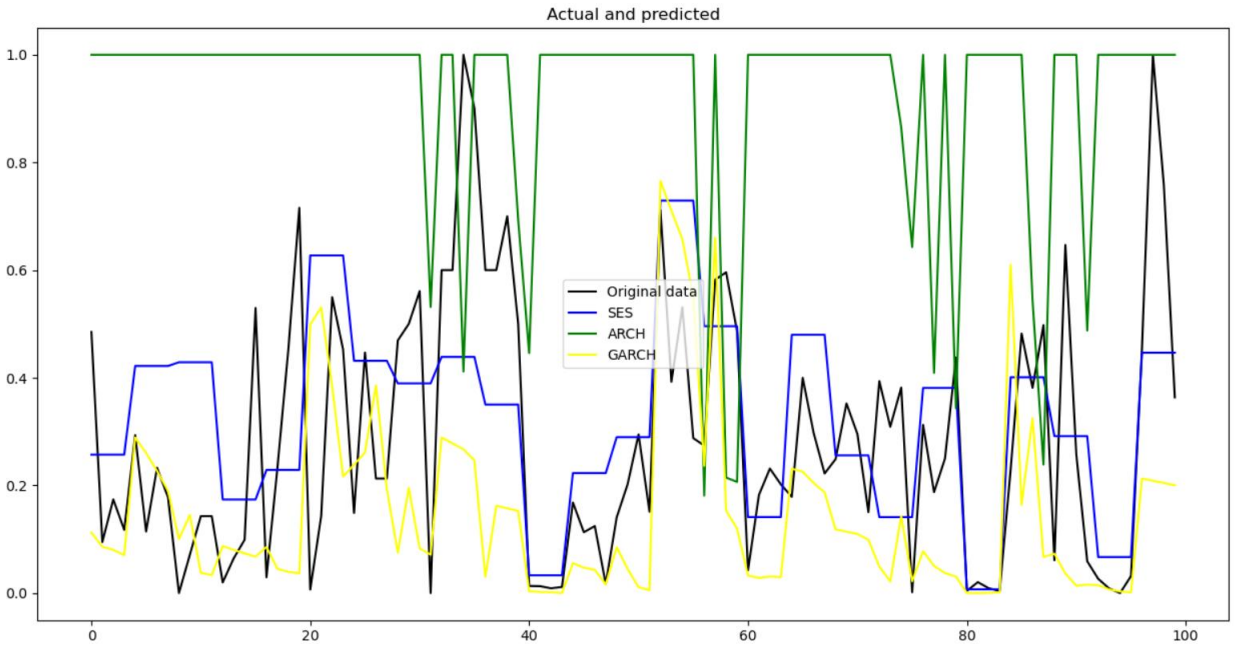


Рис 3.3 График прогнозов моделей SES, ARCH и GARCH для 25 рядов.

MLP: Многослойная нейронная сеть зависит от многих параметров: количества скрытых слоев, количества нейронов в них, выбора ошибки, функции активации для каждого скрытого слоя и количества эпох при обучении. Подобрав параметры с помощью просмотра средней точности при каждом выборе, получилось следующее: количество скрытых слоев – 3, первый скрытый слой содержит 64 нейрона, второй – 32 нейрона, третий – 16. Во всех скрытых слоях функция активации LeakyReLu. Выходной слой должен содержать функцию активации, которая выдает значения в промежутке [0;1], т.к. значения наблюдений во временных рядах лежат в этом промежутке. Для этого была выбрана функция sigmoid. Ошибка-среднеквадратическая. Количество эпох – 100. При увеличении количества эпох точность падает из-за переобучения сети, уменьшение также не дает лучшего результата.

При каждом обучении нейронной сети при одинаковых параметрах веса все равно подбираются по-разному. Из-за этого нужно сделать несколько наблюдений и взять средний результат, чтобы показать эффективность сети. Она обучалась 10 раз на каждом разбиении тренировочной и тестовой выборок: 8:2 и 9:1. Ниже приведены результаты:

При первом разбиении: 58.02%, 58.92%, 56.78%, 57.04%, 57.06%, 57.45%, 56.87%, 56,59%, 57.44%, 56,97%. Средняя точность – 57.31%.

При втором разбиении: 59.06%, 58.62%, 60.31%, 59.1%, 58.96%, 59.01%, 58.5%, 58.72%, 58.69%, 59.21%. Средняя точность – 59.02%.

Результат меньше, чем у ARIMA, даже при разделении 9:1. Пока что ARIMA остается оптимальным вариантом.

LSTM: Проверка модели проходила аналогично прошлой. Были подобраны следующие параметры: 2 скрытых LSTM слоя, первый содержит 128 нейронов, второй – 64, функции активации те же: LeakyReLu для скрытых слоев и sigmoid для выходного. Ошибка-среднеквадратическая. Количество эпох – 100, по тем же причинам. Ниже приведены значения средней точности прогноза:

При первом разбиении: 58.14%, 57,73%, 58.18%, 58.25%, 58.59%, 57.06%, 58.57%, 57,2%, 58,41, 56.82%. Средняя точность – 57.9%.

При втором разбиении: 60.63%, 60.08%, 59,63%, 60.3%, 60.62%, 60.37%, 60.69%, 59.98%, 60.37%, 60.22%. Средняя точность – 60.29%.

Рекуррентная нейронная сеть LSTM показала лучше результат, чем многослойный перцептрон, но все равно не догнала ARIMA.

CNN: У сверточной сети немного другие параметры: количество ядер, их размер, при этом функции активации, ошибка и количество эпох остаются. Было подобрано следующее: количество ядер – 64, размер одного ядра – 20, функции активации – LeakyReLu и sigmoid, среднеквадратическая ошибка и 250 эпох. Ниже приведены результаты прогнозов:

При первом разбиении: 58.2%, 58.58%, 58.39%, 58.37%, 58.35%, 58.88%, 58.39%, 59.03%, 59.05%, 58.55%. Средняя точность – 58.58%.

При втором разбиении: 60.21%, 60.41%, 60.55%, 61.34%, 60.6%, 60.52%, 60.62%, 60.77%, 60.57%, 59.81%. Средняя точность – 60.54%.

Результат лучше, чем у других нейронных сетей, но аналогично им, модель проигрывает ARIMA.

Ниже приведены графики фактических и прогнозируемых значений у 25 рядов для каждой нейронной сети и каждого разбиения (Рис 3.4-3.5):

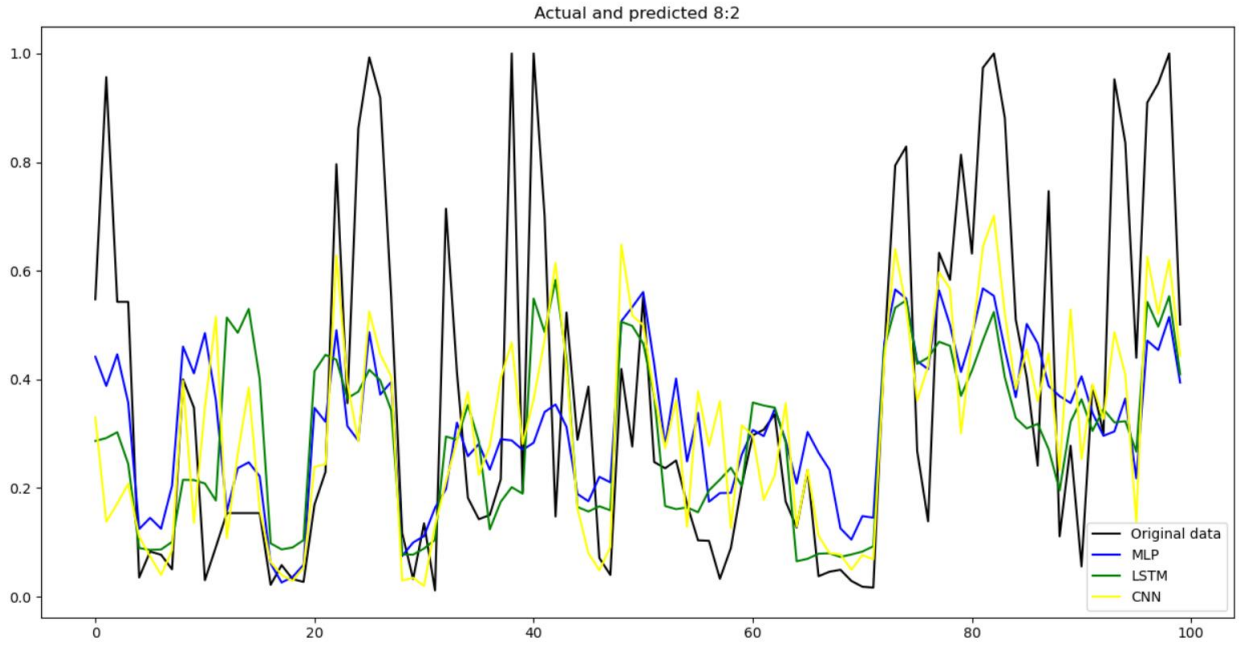


Рис 3.4 График прогнозов нейронных сетей MLP, LSTM и CNN для 25 рядов при разбиении 8:2.

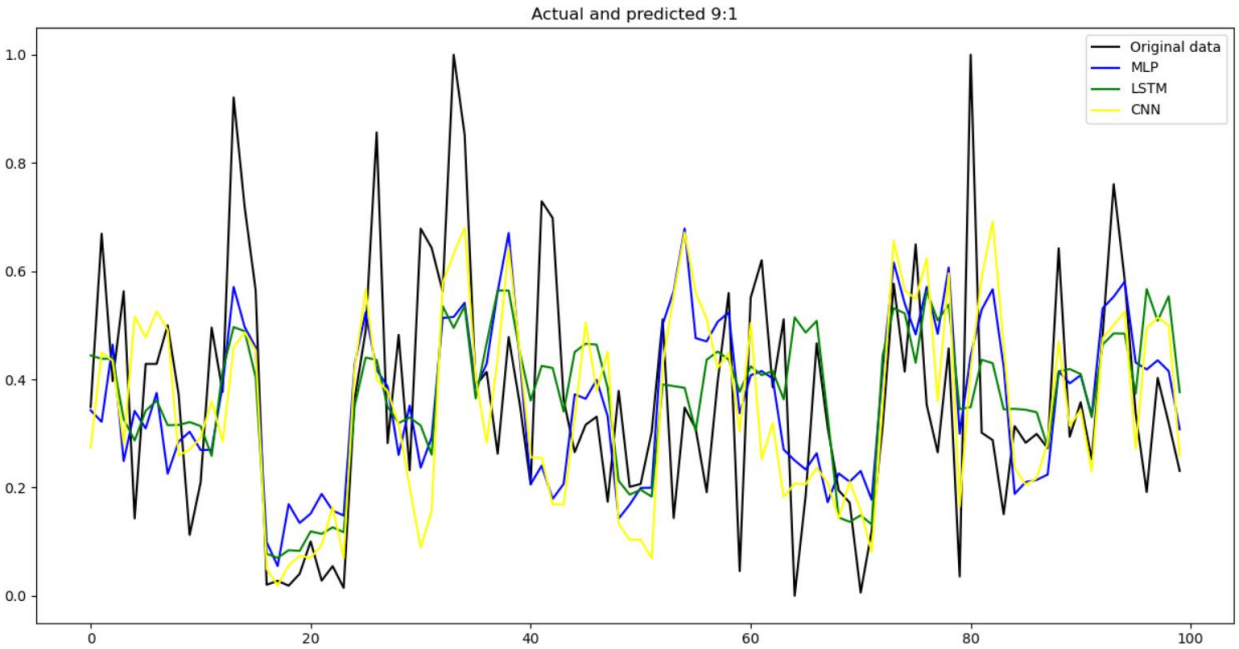


Рис 3.5 График прогнозов нейронных сетей MLP, LSTM и CNN для 25 рядов при разбиении 9:1.

3.2 Общий результат.

В этой таблице приведены результаты каждой модели в порядке убывания (Табл. 1). Как видно, лучшей моделью оказалась CNN, данную модель можно считать оптимальной для прогнозирования временных рядов в условиях небольших исходных данных. Среди моделей, не являющихся нейронными сетями лучшей стала ARIMA, она оптимальна, если нужно прогнозировать все ряды и нет возможности разбить их на выборки.

Модель	Средняя точность прогноза	
CNN	58.58%	60.54%
LSTM	57.9%	60.29%
MLP	57.31%	59.02%
ARIMA	58.31%	
SES	57.84%	
ARMA	56.26%	
AR	55.13%	
VAR	55.11%	
Naïve	54.15%	
MA	52.23%	
GARCH	38.42%	
ARCH	36.26%	

Табл. 1 Таблица средней точности моделей прогнозирования

Заключение

В рамках данной работы был проведен экспериментальный анализ работы 12-ти самых часто используемых моделей прогнозирования рядов с точки зрения небольших исходных данных, эти модели были реализованы и протестированы на большом количестве рядов небольшой длины, каждой модели были подобраны параметры, при которых они показывают лучший результат прогноза. По результатам анализа работы всех рассмотренных моделей было выявлено, что наилучшая модель для прогнозирования временных рядов малой длины для представленных исходных данных является сверточная нейронная сеть (CNN), при этом, если ставится задача прогнозирования всех временных рядов, без возможности их разбиения на выборки, то наилучшей моделью для прогнозирования является интегрированная модель авторегрессии — скользящего среднего (ARIMA).

Список литературы

1. “Introduction to Time Series and Forecasting, Second Edition” Peter J. Brockwell and Richard A. Davis, p. 7-9
2. “Forecasting: Principles and Practice” Rob J Hyndman and George Athanasopoulos, Some practical forecasting issues ch. 12.1-12.7
3. “An overview of time series forecasting models” Davide Burba
4. “Econometrics of Short and Unreliable Time Series” Thomas Url and Andreas Wörgötter, ch.III
5. “Time series analysis: forecasting and control, Fourth Edition” George E. P. Box, Gwilym M. Jenkins and Gregory C. Reinsel, part II
6. “Deep Learning for Time Series Forecasting” Jason Brawlee
7. “Forecasting: Principles and Practice” Rob J Hyndman and George Athanasopoulos, Naïve method ch. 3.1
8. “How to best understand the Naïve Forecast” Shaun Snapp
9. “STAT 501: Regression methods” Dr. Iain Pardoe, lesson 14.1
10. “Applied Time Series Analysis for Fisheries and Environmental Sciences” E. E. Holmes, M. D. Scheuerell, and E. J. Ward, ch. 4.7-4.9
11. “Statistical forecasting: notes on regression and time series analysis” Robert Nau, Introduction to ARIMA: nonseasonal models
12. “Introduction to Econometrics with R” Christoph Hanck, Martin Arnold, Alexander Gerber and Martin Schmelzer, ch.16.1
13. “Statistical forecasting: notes on regression and time series analysis” Robert Nau, Brown's Simple Exponential Smoothing (exponentially weighted moving average)
14. “Introduction to ARCH & GARCH models” Roberto Perrelli
15. “Как работает нейронная сеть: алгоритмы, обучение, функции активации и потери” Станислав Исаков
16. “Understanding LSTM Networks” Christopher Olah
17. “Как работает сверточная нейронная сеть: архитектура, примеры, особенности” Станислав Исаков

18.“KPSS Test for Stationarity” Selva Prabhakaran

Приложение

Программная реализация моделей прогнозирования на языке Python.

Naïve:

```
import csv
import matplotlib.pylab as plt
True_values = []
Predicted_values = []
Acc_list = []
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    data.append(float(row[2]))
                counter += 1
        # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            # Прогноз
            prediction = data[len(data) - i - 2]
            True_values.append(data[len(data) - i - 1])
            Predicted_values.append(prediction)
            # Вычисление точности прогноза
            if float(data[len(data) - i - 1]) != 0 and prediction != 0:
                acc = min(data[len(data) - i - 1], prediction)/max(data[len(data) - i
- 1], prediction)
            else:
                acc = 1
            Acc_list.append(acc)
            # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
            data[len(data) - i - 1] = prediction
# Вычисление средней точности прогноза
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

AR:

```
from statsmodels.tsa.ar_model import AutoReg
import csv
import matplotlib.pylab as plt
True_values = []
Predicted_values = []
Acc_list = []
# Выбор параметра p
p = 1
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    data.append(float(row[2]))
                counter += 1
        # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            try:
                # Создание модели
                model = AutoReg(data[0:len(data) - i - 1], lags=p)
                model_fit = model.fit()
                # Прогноз
                prediction = model_fit.predict(len(data[0:len(data) - i - 1]),
len(data[0:len(data) - i - 1]))
                # Модификация для интервала [0;1]
                if prediction[0] > 1:
                    prediction[0] = float(1)
                if prediction[0] < 0:
                    prediction[0] = float(0)
                True_values.append(data[len(data) - i - 1])
                Predicted_values.append(prediction[0])
                # Вычисление точности прогноза
                if float(data[len(data) - i - 1]) == 0 and prediction == 0:
                    acc = 1
                else:
                    acc = min(data[len(data) - i - 1], prediction) /
max(data[len(data) - i - 1], prediction)
                Acc_list.append(acc)
                # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                data[len(data) - i - 1] = prediction[0]
            # При ошибках в прогнозировании точность становится равной нулю
            except ZeroDivisionError:
                Acc_list.append(0)
                continue
            except ValueError:
                Acc_list.append(0)
                continue
# Вычисление средней точности прогноза
```

```
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

MA:

```
from statsmodels.tsa.arima_model import ARMA
import csv
import numpy
import matplotlib.pylab as plt
True_values = []
Predicted_values = []
Acc_list = []
# Выбор параметра q
q = 3
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    data.append(float(row[2]))
                counter += 1
        # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            try:
                # Создание модели
                model = ARMA(data[0:len(data) - i - 1], order=(0, q))
                model_fit = model.fit(disp=False)
                # Прогноз
                prediction = model_fit.predict(len(data[0:len(data) - i - 1]),
len(data[0:len(data) - i - 1]))
                # Модификация для интервала [0;1]
                if prediction[0] > 1:
                    prediction[0] = float(1)
                if prediction[0] < 0:
                    prediction[0] = float(0)
                True_values.append(data[len(data) - i - 1])
                Predicted_values.append(prediction[0])
                # Вычисление точности прогноза
                if float(data[len(data) - i - 1]) == 0 and prediction == 0:
                    acc = 1
                else:
                    acc = min(data[len(data) - i - 1], prediction) /
max(data[len(data) - i - 1], prediction)
                Acc_list.append(acc)
```

```
                # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                data[len(data) - i - 1] = prediction[0]
            # При ошибках в прогнозировании точность становится равной нулю
            except ValueError:
                Acc_list.append(0)
                continue
            except numpy.linalg.LinAlgError:
                Acc_list.append(0)
                continue
            except ZeroDivisionError:
                Acc_list.append(0)
                continue
# Вычисление средней точности прогноза
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

ARMA:

```
import csv
import numpy
import matplotlib.pylab as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
True_values = []
Predicted_values = []
Acc_list = []
# Выбор параметров p и q
p = 1
q = 1
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    data.append(float(row[2]))
                counter += 1
            # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            try:
                # Создание модели
                model = SARIMAX(data[0:len(data) - i - 1], order=(p, 0, q))
                model_fit = model.fit(disp=False)
                # Прогноз
                prediction = model_fit.predict(len(data[0:len(data) - i - 1]),
```

```
                                                    len(data[0:len(data) - i - 1]),
typ='levels')
                # Модификация для интервала [0;1]
                if prediction[0] > 1:
                    prediction[0] = float(1)
                if prediction[0] < 0:
                    prediction[0] = float(0)
                True_values.append(data[len(data) - i - 1])
                Predicted_values.append(prediction[0])
                # Вычисление точности прогноза
                if float(data[len(data) - i - 1]) == 0 and prediction == 0:
                    acc = 1
                else:
                    acc = min(data[len(data) - i - 1], prediction) /
max(data[len(data) - i - 1], prediction)
                Acc_list.append(acc)
                # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                data[len(data) - i - 1] = prediction[0]
            # При ошибках в прогнозировании точность становится равной нулю
            except ZeroDivisionError:
                Acc_list.append(0)
                continue
            except numpy.linalg.LinAlgError:
                Acc_list.append(0)
                continue
            except IndexError:
                Acc_list.append(0)
                continue
# Вычисление средней точности прогноза
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

ARIMA:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
import csv
import numpy
import matplotlib.pylab as plt
True_values = []
Predicted_values = []
Acc_list = []
# Выбор параметров p,d и q
p = 1
d = 1
q = 1
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
```

```
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    data.append(float(row[2]))
                counter += 1
        # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            try:
                # Создание модели
                model = SARIMAX(data[0:len(data) - i - 1], order=(p, d, q))
                model_fit = model.fit(disp=False)
                # Прогноз
                prediction = model_fit.predict(len(data[0:len(data) - i - 1]),
                                               len(data[0:len(data) - i - 1]),
typ='levels')
                # Модификация для интервала [0;1]
                if prediction[0] > 1:
                    prediction[0] = float(1)
                if prediction[0] < 0:
                    prediction[0] = float(0)
                True_values.append(data[len(data) - i - 1])
                Predicted_values.append(prediction[0])
                # Вычисление точности прогноза
                if float(data[len(data) - i - 1]) == 0 and prediction == 0:
                    acc = 1
                else:
                    acc = min(data[len(data) - i - 1], prediction) /
max(data[len(data) - i - 1], prediction)
                Acc_list.append(acc)
                # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                data[len(data) - i - 1] = prediction[0]
            # При ошибках в прогнозировании точность становится равной нулю
            except ZeroDivisionError:
                Acc_list.append(0)
                continue
            except numpy.linalg.LinAlgError:
                Acc_list.append(0)
                continue
            except IndexError:
                Acc_list.append(0)
                continue
# Вычисление средней точности прогноза
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

VAR:

```
from statsmodels.tsa.vector_ar.var_model import VAR
import csv
import matplotlib.pylab as plt
True_values = []
Predicted_values = []
Acc_list = []
# Выбор параметра p
p = 1
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    x = []
                    for i in range(2):
                        x.append(float(row[2]))
                    data.append(x)
                counter += 1
        # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            try:
                # Создание модели
                model = VAR(data[0:len(data) - i - 1])
                model_fit = model.fit(maxlags=p)
                # Прогноз
                prediction = model_fit.forecast(model_fit.y, steps=1)
                # Модификация для интервала [0;1]
                if prediction[0][0] > 1:
                    prediction[0][0] = float(1)
                if prediction[0][0] < 0:
                    prediction[0][0] = float(0)
                True_values.append(data[len(data) - i - 1][0])
                Predicted_values.append(prediction[0][0])
                # Вычисление точности прогноза
                if float(data[len(data) - i - 1][0]) == 0 and prediction[0][0] == 0:
                    acc = 1
                else:
                    acc = min(data[len(data) - i - 1][0], prediction[0][0]) /
max(data[len(data) - i - 1][0],

prediction[0][0])
                Acc_list.append(acc)
                # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                for j in range(2):
                    data[len(data) - i - 1][j] = prediction[0][0]
            # При ошибках в прогнозировании точность становится равной нулю
            except ValueError:
                Acc_list.append(0)
```

```
                continue
# Вычисление средней точности прогноза
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

SES (для одного коэффициента сглаживания):

```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
import csv
import matplotlib.pylab as plt
True_values = []
Predicted_values = []
Acc_list = []
# Выбор коэффициента сглаживания
Smooth = 0.26
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    data.append(float(row[2]))
                counter += 1
        # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            try:
                # Создание модели
                model = SimpleExpSmoothing(data[0:len(data) - i - 1])
                model_fit = model.fit(smoothing_level=Smooth)
                # Прогноз
                prediction = model_fit.predict(len(data[0:len(data) - i - 1]),
len(data[0:len(data) - i - 1]))
                # Модификация для интервала [0;1]
                if prediction[0] > 1:
                    prediction[0] = float(1)
                if prediction[0] < 0:
                    prediction[0] = float(0)
                True_values.append(data[len(data) - i - 1])
                Predicted_values.append(prediction[0])
                # Вычисление точности прогноза
                if float(data[len(data) - i - 1]) == 0 and prediction == 0:
                    acc = 1
                else:
                    acc = min(data[len(data) - i - 1], prediction) /
max(data[len(data) - i - 1], prediction)
```

```
                Acc_list.append(acc)
                # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                data[len(data) - i - 1] = prediction[0]
            # При ошибках в прогнозировании точность становится равной нулю
            except IndexError:
                Acc_list.append(0)
                continue
# Вычисление средней точности прогноза
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

SES (для проверки всех значений коэффициента сглаживания):

```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
import csv
import matplotlib.pylab as plt
All_acc_list = []
# Цикл для проверки всех значений коэффициента сглаживания от 0 до 1 с шагом 0.01
for S in range(101):
    Smooth = S/100
    Acc_list = []
    for num in range(1146):
        # Считывание входных данных
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            month_quantity = 0
            for row in reader:
                month_quantity += 1
        # Исключение рядов длиной меньше 5
        if month_quantity > 4:
            with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
                reader = csv.reader(f)
                data = []
                counter = 0
                for row in reader:
                    if 0 < counter <= month_quantity - 1:
                        data.append(float(row[2]))
                    counter += 1
            # Цикл с 4 итерациями для прогнозирования 4 значений
            for i in [3, 2, 1, 0]:
                try:
                    # Создание модели
                    model = SimpleExpSmoothing(data[0:len(data) - i - 1])
                    model_fit = model.fit(smoothing_level=Smooth)
                    # Прогноз
                    prediction = model_fit.predict(len(data[0:len(data) - i - 1]),
len(data[0:len(data) - i - 1]))
                    # Модификация для интервала [0;1]
                    if prediction[0] > 1:
                        prediction[0] = float(1)
                    if prediction[0] < 0:
                        prediction[0] = float(0)
                    # Вычисление точности прогноза
```

```
                    if float(data[len(data) - i - 1]) == 0 and prediction == 0:
                        acc = 1
                    else:
                        acc = min(data[len(data) - i - 1], prediction) /
max(data[len(data) - i - 1], prediction)
                    Acc_list.append(acc)
                    # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                    data[len(data) - i - 1] = prediction[0]
                # При ошибках в прогнозировании точность становится равной нулю
                except IndexError:
                    Acc_list.append(0)
                    continue
    # Вычисление средней точности прогноза
    Average_Acc = sum(Acc_list)/len(Acc_list)
    All_acc_list.append(Average_Acc)
# Нахождение максимальной средней точности и коэффициента сглаживания дающего эту
точность
M = max(All_acc_list)
for i in range (len(All_acc_list)):
    if All_acc_list[i] == M:
        best = i
print(M)
print(best)
# График зависимости средней точности от коэффициента сглаживания
plt.plot(All_acc_list, color='blue', label = 'Accuracy')
plt.legend(loc='best')
plt.title('Average accuracy for different smoothing level')
plt.show()
```

ARCH:

```
from arch import arch_model
import csv
import matplotlib.pylab as plt
True_values = []
Predicted_values = []
Acc_list = []
# Выбор параметра p
p = 1
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    data.append(float(row[2]))
                counter += 1
        # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            try:
```

```
                # Создание модели
                model = arch_model(data[0:len(data) - i - 1], mean='Zero',
vol='ARCH', p=p, rescale=True)
                model_fit = model.fit()
                # Прогноз
                prediction_0 = model_fit.forecast(horizon=1)
                prediction =
prediction_0.variance.values[len(prediction_0.variance.values) - 1][0]
                # Модификация для интервала [0;1]
                if prediction > 1:
                    prediction = float(1)
                if prediction < 0:
                    prediction = float(0)
                True_values.append(data[len(data) - i - 1])
                Predicted_values.append(prediction)
                # Вычисление точности прогноза
                if float(data[len(data) - i - 1]) == 0 and prediction == 0:
                    acc = 1
                else:
                    acc = min(data[len(data) - i - 1], prediction) /
max(data[len(data) - i - 1], prediction)
                Acc_list.append(acc)
                # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                data[len(data) - i - 1] = prediction
            # При ошибках в прогнозировании точность становится равной нулю
            except ValueError:
                Acc_list.append(0)
                continue
# Вычисление средней точности прогноза
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

GARCH:

```
from arch import arch_model
import csv
import matplotlib.pylab as plt
True_values = []
Predicted_values = []
Acc_list = []
# Выбор параметров p и q
p = 1
q = 1
for num in range(1146):
    # Считывание входных данных
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    # Исключение рядов длиной меньше 5
    if month_quantity > 4:
        with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
```

```
            reader = csv.reader(f)
            data = []
            counter = 0
            for row in reader:
                if 0 < counter <= month_quantity - 1:
                    data.append(float(row[2]))
                counter += 1
        # Цикл с 4 итерациями для прогнозирования 4 значений
        for i in [3, 2, 1, 0]:
            try:
                # Создание модели
                model = arch_model(data[0:len(data) - i - 1], mean='Zero',
vol='GARCH', p=p, q=q, rescale=False)
                model_fit = model.fit()
                # Прогноз
                prediction_0 = model_fit.forecast(horizon=1)
                prediction =
prediction_0.variance.values[len(prediction_0.variance.values) - 1][0]
                # Модификация для интервала [0;1]
                if prediction > 1:
                    prediction = float(1)
                if prediction < 0:
                    prediction = float(0)
                True_values.append(data[len(data) - i - 1])
                Predicted_values.append(prediction)
                # Вычисление точности прогноза
                if float(data[len(data) - i - 1]) == 0 and prediction == 0:
                    acc = 1
                else:
                    acc = min(data[len(data) - i - 1], prediction) /
max(data[len(data) - i - 1], prediction)
                Acc_list.append(acc)
                # Изменение значения наблюдения на прогнозируемое для дальнейшего
прогноза следующих наблюдений
                data[len(data) - i - 1] = prediction
            # При ошибках в прогнозировании точность становится равной нулю
            except ValueError:
                Acc_list.append(0)
                continue
            except ZeroDivisionError:
                Acc_list.append(0)
                continue
# Вычисление средней точности прогноза
Average_Acc = sum(Acc_list)/len(Acc_list)
print(Average_Acc)
# График фактических и прогнозируемых значений наблюдений
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
```

MLP:

```
import csv
import numpy as np
import matplotlib.pylab as plt
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers.advanced_activations import *
```

```
from keras.callbacks import ReduceLROnPlateau
from keras.optimizers import Adam
data = []
# Считывание всех входных данных
for num in range(1146):
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        one_data = []
        counter = 0
        for row in reader:
            if 0 < counter <= month_quantity - 1:
                one_data.append(float(row[2]))
            counter += 1
        # Исключение рядов длиной меньше 5
        if len(one_data) > 4:
            data.append(one_data)
# Разбиение на выборки
train, test = ((data[0:int(len(data) * 0.8)], data[int(len(data) * 0.8) + 1:]))
# Заполнение рядов нулями в их начале для того, чтобы входной слой сети был полностью
заполнен
for i in range(len(train)):
    for j in range(len(train[i]), 47):
        train[i].insert(0, 0)
for i in range(len(test)):
    for j in range(len(test[i]), 47):
        test[i].insert(0, 0)
train = np.array(train)
test = np.array(test)
# Отведение 4 последних наблюдений на прогнозирование
X_train = train[:, 0:-4]
X_test = test[:, 0:-4]
Y_train = train[:, -4:]
Y_test = test[:, -4:]
# Создание слоев сети
model = Sequential()
model.add(Dense(64, input_dim=43))
model.add(LeakyReLU())
model.add(Dense(32))
model.add(LeakyReLU())
model.add(Dense(16))
model.add(LeakyReLU())
model.add(Dense(4))
model.add(Activation('sigmoid'))
opt = Adam(lr=0.001)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=5,
min_lr=0.000001, verbose=1)
# Компилирование сети
model.compile(optimizer=opt,
              loss='mse',
              metrics=['accuracy'])
# Обучение сети
history = model.fit(X_train, Y_train,
          epochs=100,
          batch_size=512,
          verbose=2,
          validation_data=(X_train, Y_train),
```

```
          shuffle=True,
          callbacks=[reduce_lr])
# График фактических и прогнозируемых значений наблюдений
Predicted_values = model.predict(np.array(X_test))
True_values = Y_test
Predicted_values = Predicted_values.reshape(Predicted_values.shape[0] *
Predicted_values.shape[1])
True_values = True_values.reshape(True_values.shape[0] * True_values.shape[1])
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
# Вычисление средней точности прогноза
True_values[True_values == 0] = 1e-10
Predicted_values[Predicted_values == 0] = 1e-10
tmp = True_values/Predicted_values
tmp[tmp > 1] = 1/tmp[tmp > 1]
print(np.sum(tmp)/(Predicted_values.shape))
```

LSTM:

```
import csv
import numpy as np
import matplotlib.pylab as plt
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.layers.advanced_activations import *
from keras.callbacks import ReduceLROnPlateau
from keras.optimizers import Adam
data = []
# Считывание всех входных данных
for num in range(1146):
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        one_data = []
        counter = 0
        for row in reader:
            if 0 < counter <= month_quantity - 1:
                one_data.append(float(row[2]))
            counter += 1
        # Исключение рядов длиной меньше 5
        if len(one_data) > 4:
            data.append(one_data)
# Разбиение на выборки
train, test = ((data[0:int(len(data) * 0.8)], data[int(len(data) * 0.8) + 1:]))
# Заполнение рядов нулями в их начале для того, чтобы входной слой сети был полностью
заполнен
for i in range(len(train)):
    for j in range(len(train[i]), 47):
        train[i].insert(0, 0)
for i in range(len(test)):
    for j in range(len(test[i]), 47):
        test[i].insert(0, 0)
train = np.array(train)
```

```
test = np.array(test)
# Отведение 4 последних наблюдений на прогнозирование
X_train = train[:, 0:-4]
X_test = test[:, 0:-4]
Y_train = train[:, -4:]
Y_test = test[:, -4:]
# Изменение размерности данных, т.к. в LSTM принимаются трехмерные массивы
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
# Создание слоев сети
model = Sequential()
model.add(LSTM(128, input_shape=(43, 1), return_sequences=True))
model.add(LeakyReLU())
model.add(LSTM(64))
model.add(LeakyReLU())
model.add(Dense(4, activation='sigmoid'))
opt = Adam(lr=0.001)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=5,
min_lr=0.000001, verbose=1)
# Компилирование сети
model.compile(optimizer=opt,
              loss='mse',
              metrics=['accuracy'])
# Обучение сети
history = model.fit(X_train, Y_train,
          epochs=80,
          batch_size=256,
          verbose=2,
          validation_data=(X_train, Y_train),
          shuffle=True,
          callbacks=[reduce_lr]
          )
# График фактических и прогнозируемых значений наблюдений
Predicted_values = model.predict(np.array(X_test))
True_values = Y_test
Predicted_values = Predicted_values.reshape(Predicted_values.shape[0] *
Predicted_values.shape[1])
True_values = True_values.reshape(True_values.shape[0] * True_values.shape[1])
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
# Вычисление средней точности прогноза
True_values[True_values == 0] = 1e-10
Predicted_values[Predicted_values == 0] = 1e-10
tmp = True_values/Predicted_values
tmp[tmp > 1] = 1/tmp[tmp > 1]
print(np.sum(tmp)/(Predicted_values.shape))
```

CNN:

```
import csv
import numpy as np
import matplotlib.pylab as plt
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.callbacks import ReduceLROnPlateau
from keras.optimizers import Adam
from keras.layers.convolutional import Conv1D
```

```
data = []
# Считывание всех входных данных
for num in range(1146):
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        month_quantity = 0
        for row in reader:
            month_quantity += 1
    with open('Time_Series/item' + str(1213 + num) + '.csv') as f:
        reader = csv.reader(f)
        one_data = []
        counter = 0
        for row in reader:
            if 0 < counter <= month_quantity - 1:
                one_data.append(float(row[2]))
            counter += 1
        # Исключение рядов длиной меньше 5
        if len(one_data) > 4:
            data.append(one_data)
# Разбиение на выборки
train, test = ((data[0:int(len(data) * 0.8)], data[int(len(data) * 0.8) + 1:]))
# Заполнение рядов нулями в их начале для того, чтобы входной слой сети был полностью
заполнен
for i in range(len(train)):
    for j in range(len(train[i]), 47):
        train[i].insert(0, 0)
for i in range(len(test)):
    for j in range(len(test[i]), 47):
        test[i].insert(0, 0)
train = np.array(train)
test = np.array(test)
# Отведение 4 последних наблюдений на прогнозирование
X_train = train[:, 0:-4]
X_test = test[:, 0:-4]
Y_train = train[:, -4:]
Y_test = test[:, -4:]
# Изменение размерности данных, т.к. в CNN принимаются трехмерные массивы
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
# Создание слоев сети
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=20, activation='relu', input_shape=(43, 1)))
model.add(Flatten())
model.add(Dense(4, activation='sigmoid'))
opt = Adam(lr=0.001)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=5,
min_lr=0.000001, verbose=1)
# Компилирование сети
model.compile(optimizer=opt,
              loss='mse',
              metrics=['accuracy'])
# Обучение сети
history = model.fit(X_train, Y_train,
          epochs=250,
          batch_size=256,
          verbose=2,
          validation_data=(X_train, Y_train),
          shuffle=True,
          callbacks=[reduce_lr]
          )
# График фактических и прогнозируемых значений наблюдений
```

```
Predicted_values = model.predict(np.array(X_test))
True_values = Y_test
Predicted_values = Predicted_values.reshape(Predicted_values.shape[0] *
Predicted_values.shape[1])
True_values = True_values.reshape(True_values.shape[0] * True_values.shape[1])
plt.plot(True_values, color='black', label = 'Original data')
plt.plot(Predicted_values, color='blue', label = 'Predicted data')
plt.legend(loc='best')
plt.title('Actual and predicted')
plt.show()
# Вычисление средней точности прогноза
True_values[True_values == 0] = 1e-10
Predicted_values[Predicted_values == 0] = 1e-10
tmp = True_values/Predicted_values
tmp[tmp > 1] = 1/tmp[tmp > 1]
print(np.sum(tmp)/(Predicted_values.shape))
```