Математические структуры и моделирование
2025. № 3 (75). С. 101–115

УДК 51-77
DOI 10.24147/2222-8772.2025.3.101-115

МОДЕЛЬ ПРОГНОЗИРОВАНИЯ ВРЕМЕННЫХ РЯДОВ НА ОСНОВЕ СИНГУЛЯРНО-СПЕКТРАЛЬНОГО АНАЛИЗА И НЕЙРОННОЙ СЕТИ LSTM

А.Д. Байдалин
аспирант, e-mail: alexander.baydalin@gmail.com
В.В. Варламов
д.ф.-м.н., профессор, e-mail: varlamov@subsiu.ru

Сибирский государственный индустриальный университет, Новокузнецк, Россия

Аннотация. Статья посвящена разработке модели прогнозирования цены акции компании с использованием сингулярно-спектрального анализа в комбинации с нейронной сетью LSTM. В статье предлагается подход к анализу временных рядов цен акций, основанный на применении сингулярно-спектрального анализа, который позволяет выделить скрытые закономерности и тренды в данных, а также устранить шум, улучшая точность прогноза. Методика включает в себя разложение временного ряда на несколько составляющих, что способствует более глубокому пониманию динамики цен и повышению надежности предсказаний. Результаты экспериментов показали высокую эффективность предложенного метода по сравнению с традиционными статистическими и машинными методами анализа, что открывает новые возможности для оптимизации инвестиционных стратегий и анализа финансовых рынков.

Ключевые слова: сингулярно-спектральный анализ, временные ряды, финансовые рынки, анализ данных, инвестиционные стратегии, нейронные сети.

Введение

Сингулярно-спектральный анализ (SSA, Singular Spectrum Analysis) является мощным методом для анализа временных рядов, который активно используется в различных областях науки и техники. В отличие от традиционных статистических методов, таких как автокорреляция или спектральный анализ, SSA не требует строгих предположений о распределении данных или линейности временного ряда. Это делает его особенно полезным для работы с нестационарными, шумными и сложными данными, часто встречающимися в реальных приложениях.

Основная идея SSA заключается в разложении временного ряда на компоненты, отражающие его скрытые структуры – тренды, циклы и случайные флуктуации. Процесс анализа включает несколько этапов: создание матрицы траектории, сингулярное разложение, выбор значимых компонент и их реконструкция. Это позволяет не только выявить долгосрочные тренды и циклические изменения, но и очистить данные от шума, что важно для предсказания и прогнозирования.

Метод SSA нашёл широкое применение в таких областях, как экономика, финансы, экология, метеорология и другие, где требуется анализ временных рядов для выявления скрытых закономерностей и принятия информированных решений. Кроме того, SSA активно используется для обработки и анализа сигналов в физике, биологии и инженерии. В статье рассматриваются основные принципы и этапы SSA, а также его применение для решения реальных задач, связанных с обработкой временных рядов.

В данной работе применяется метод анализа временных рядов SSA, который позволяет эффективно выделить основные компоненты исходных данных и устранить шум. Далее для построения прогноза на основе полученных временных рядов используется модель LSTM (Long Short-Term Memory), способная учитывать долгосрочные зависимости и тенденции в данных. Применение LSTM в сочетании с SSA позволяет улучшить точность прогноза, обеспечив более стабильные результаты, по сравнению с традиционными методами. Таким образом, предложенный подход демонстрирует высокую эффективность в прогнозировании временных рядов, что открывает возможности для его использования в различных областях, требующих точного анализа и предсказания динамики данных.

1. Основные этапы метода SSA

Применение SSA к котировкам акций представляет собой метод анализа временных рядов, который позволяет выявлять скрытые закономерности, тренды и цикличности в данных о ценах акций. SSA – это метод, который используется для выделения структуры временных рядов без необходимости использования сложных моделей или предварительных предположений о данных. В этом параграфе кратко рассмотрим основные этапы метода SSA.

1.1. Представление временного ряда в виде матрицы траектории

Первым этапом SSA является преобразование временного ряда в матричную форму. Это необходимо для того, чтобы последующий процесс сингулярного разложения (SVD) мог выделить скрытые структуры и компоненты ряда. Этот этап называется формированием матрицы траектории. Рассмотрим его более детально [1].

Предположим, что у нас есть одномерный временной ряд $\{X_1, X_2, \ldots, X_n\}$, состоящий из N наблюдений. Это могут быть данные о температуре, стоимости акций, показателях продаж и т. д. В данной статье будет рассматриваться реализация SSA по отношению к временному ряду, состоящему из истории котировок акций компании «Сбербанк».

Для того чтобы сформировать матрицу траектории, используется принцип скользящего окна. В SSA выбирается фиксированная длина окна L, которое «перемещается» по временным данным, формируя из них подматрицы. Каждое окно охватывает L последовательных значений временного ряда, и это окно переносится по ряду с некоторым шагом d. Обычно шаг d равен 1, но в некоторых случаях могут использоваться и другие значения.

Если окно длины L и шаг $d = 1$, то для первого окна мы возьмём значения временного ряда с индексами $1, 2, \ldots, L$; для второго окна значения с индексами $2, 3, \ldots, L + 1$; и т. д. Таким образом, для временного ряда длиной N получится $M = N - L + 1$ окон [2].

Когда окно определено, необходимо из этих окон собрать матрицу траектории. Каждое окно представляется как строка в матрице. Таким образом, получаем матрицу размера $L * M$, где L — количество строк (размер окна); $M = N–L + 1$ – количество столбцов (количество окон). Каждая строка в этой матрице представляет собой последовательность значений временного ряда в пределах одного окна, а каждый столбец – это окно, смещённое по временной оси.

Размер окна L играет важную роль в SSA. Если окно слишком большое, то может быть захвачено слишком много информации, что приведёт к затруднениям в выделении нужных компонент. Если окно слишком маленькое, то может быть упущена информация о долгосрочных трендах и циклах.

Траекторная матрица временного ряда X состоит из векторов вложения в качестве столбцов. Траекторная матрица представлена в следующей формуле:

$$X = (f_{ij})_{i,j=1}^{L,K} = \begin{bmatrix} f_0 & \cdots & f_{K-1} \\ \vdots & \ddots & \vdots \\ f_{L-1} & \cdots & f_{N-1} \end{bmatrix}. \tag{1}$$

Пусть $N = 6$, т. е. у нас есть временной ряд из шести значений: $\{X_1, X_2, X_3, X_4, X_5, X_6\}$. Пусть длина окна $L = 3$. Тогда матрица траекторий будет иметь вид:

$$\begin{bmatrix} X_1 & X_2 & X_3 \\ X_2 & X_3 & X_4 \\ X_3 & X_4 & X_5 \\ X_4 & X_5 & X_6 \end{bmatrix}.$$

Представленная матрица уже является входными данными для SSA. Задача SSA заключается в том, чтобы найти скрытые зависимости и структуры в этом представлении данных. Траекторная матрица является базой для реализации следующего этапа метода SSA: сингулярного разложения (SVD, Singular Value Decomposition), которое позволяет выделить важные компоненты, такие как тренды, циклы и случайные флуктуации.

1.2. SVD траекторной матрицы

SVD – это ключевая операция в SSA, которая позволяет выделить основные компоненты временного ряда, представленного в виде матрицы траектории. SVD используется для разложения матрицы на несколько составных частей [1].

SVD матрицы A размером $L \times M$ (где L – количество строк, а M – количество столбцов) представляет собой разложение этой матрицы на три матрицы:

$$A = U\Sigma V^T, \tag{2}$$

где U – матрица размером $L \times L$, содержащая левые сингулярные векторы; Σ – диагональная матрица размером $L \times M$ с сингулярными числами на диагонали, которые являются квадратными корнями собственных значений матрицы $A^T A$; V^T – транспонированная матрица размером $M \times M$, содержащая правые сингулярные векторы.

Сингулярные числа (элементы диагональной матрицы Σ) представляют собой показатели важности каждого из компонентов матрицы A. Чем больше сингулярное число, тем более значимый компонент оно представляет. Сингулярные векторы в матрицах U и V содержат информацию о направлениях, в которых эти компоненты проявляются в исходных данных.

Так, для траекторной матрицы A размером $L \times M$ применение SVD даёт следующие компоненты:

- Левые сингулярные векторы U описывают, как компоненты (тренды, циклы и шум) связаны с наблюдаемыми значениями временного ряда на каждом моменте времени. Эти векторы определяют веса, с которыми каждая компонента влияет на наблюдение в определённые моменты времени.
- Сингулярные числа Σ представляют собой величины, указывающие на силу каждой компоненты. Большие сингулярные числа означают более значимую компоненту, которая будет более выражена в данных.
- Правые сингулярные векторы V^T показывают, как каждая компонента связана с временными интервалами. Эти векторы описывают, в каком порядке и с какой интенсивностью различные компоненты проявляются в каждом из окон временного ряда [3].

1.3. Реконструкция ряда

SVD позволяет выделить несколько важных аспектов данных. Если сингулярные числа σ_i быстро убывают (например, несколько из них значительно больше остальных), это означает, что данные можно аппроксимировать с использованием только первых нескольких компонент. Эти компоненты обычно содержат информацию о долгосрочных трендах или циклических паттернах в данных. Меньшие сингулярные числа связаны с компонентами, которые можно интерпретировать как шум или случайные колебания. Обычно компоненты с малыми сингулярными числами можно отбросить, поскольку они не вносят существенного вклада в основную структуру данных.

Реконструкция осуществляется с использованием тех же сингулярных векторов, но с отбором только значимых компонент. Это позволяет выделить только важные паттерны, такие как тренды или циклические колебания, и отфильтровать менее значимые компоненты, например, шум.

Реконструкция происходит для каждой компоненты временного ряда отдельно:

- Если SVD показало, что первая компонента представляет собой тренд, то её можно восстановить и использовать для анализа долгосрочных изменений.
- Если вторая компонента представляет собой циклические колебания, то она также восстанавливается, и мы можем изучить частотные или сезонные паттерны в данных.

- Меньшие компоненты, связанные с шумом, часто отбрасываются, так как они не содержат значимой информации.

Чтобы восстановить основные компоненты временного ряда, мы выбираем $r = 2$ самых значимых компонент, что соответствует первому и второму сингулярным числам. Оставшиеся компоненты (с малыми сингулярными числами) можно отбросить как шум.

Реконструированная матрица будет выглядеть так (см. формулу (2)):

$$A_r = U_r \Sigma_r V_r^T,$$

где U_r, Σ_r, V_r^T содержат только первые два столбца, два сингулярных числа и две строки соответственно. В результате мы получим новую матрицу траектории, которая воссоздаст временной ряд с исключением шума.

2. Рекуррентная нейронная сеть LSTM

Когда результаты SSA применяются к данным, нейронные сети могут использовать эти очищенные и разделённые компоненты для более точного анализа и предсказания. Разделив временной ряд на тренды, циклические компоненты и остаточные элементы, можно значительно упростить задачу, предоставив нейронной сети более понятные и чётко структурированные входные данные. Это позволяет модели сосредоточиться на самых значимых паттернах и значительно повышает её способность делать точные прогнозы.

Таким образом, комбинирование методов SSA и нейронных сетей представляет собой перспективный подход для работы с временными рядами. Эта комбинация открывает новые возможности для повышения точности предсказаний, особенно в задачах, где данные имеют высокую степень сложности и зашумлённости.

Рекуррентные нейронные сети (RNN, Recurrent Neural Networks) – это класс нейронных сетей, предназначенных для обработки последовательных данных, таких как временные ряды, тексты, звуковые сигналы и другие типы последовательностей. В отличие от стандартных нейронных сетей (например, многослойных перцептронов), в RNN имеется механизм, который позволяет учитывать информацию о предыдущих шагах (временных точках) последовательности.

Основная идея RNN заключается в наличии петли связи, которая позволяет сети «помнить» информацию о предыдущих состояниях. Это делает RNN идеальными для работы с последовательными данными, где контекст предыдущих элементов важен для понимания текущего.

В отличие от обычных нейронных сетей, где данные передаются от слоя к слою, в RNN выход на текущем шаге зависит от выходных данных на предыдущем шаге. Это создаёт «память» о том, что происходило раньше.

В самой простой форме RNN представляет собой цепочку нейронов, каждый из которых соединён с собой через петлю. Рассмотрим основные элементы этого процесса.

Пусть у нас есть последовательность данных (например, восстановленный ряд после SSA)

$$x_1,\ x_2,\ x_3,\ \ldots,\ x_T.$$

Для каждого времени t на вход в RNN подаётся вектор x_t, и сеть генерирует скрытое состояние h_t на основе предыдущего скрытого состояния h_{t-1} и текущего входа x_t. Процесс вычислений для RNN на каждом временном шаге можно описать следующим уравнением [4]:

$$h_t = f(W_h h_{t-1} + W_t x_t + b), \tag{3}$$

где

h_t – скрытое состояние на текущем шаге t;

h_{t-1} – скрытое состояние на предыдущем шаге $t-1$;

x_t – входной вектор на текущем шаге t;

W_h, W_t – весовые матрицы для скрытого состояния входа;

b – вектор смещения;

$f()$ – нелинейная активация, часто используемая – это функция tanh или ReLU.

Выход из сети осуществляется по следующей формуле:

$$y_t = W_y h_t + c, \tag{4}$$

где y_t – выход сети на текущем шаге t; W_y - матрица весов для выхода; c – вектор смещения на выходе.

Таким образом, на каждом шаге t скрытое состояние h_t зависит от текущего входа x_t и скрытого состояния h_{t-1}, которое несёт информацию о прошлых шагах. Это позволяет модели «запоминать» контекст последовательности.

LSTM (Long Short-Term Memory) – это улучшенный тип RNN, который решает основные проблемы стандартных RNN, такие как затухание и взрыв градиентов. LSTM включает в себя специальные ячейки памяти и несколько гейтов для контроля потока информации, что позволяет ей эффективно запоминать долгосрочные зависимости в данных.

Забывающее состояние (Forget gate): решает, какую информацию из предыдущего состояния памяти C_{t-1} нужно забыть. Это делается с помощью сигмоидной функции активации σ, которая выводит значения от 0 до 1, где 0 означает «забыть всё», а 1 – «сохранить всё». Забывающее состояние определяется формулой

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f), \tag{5}$$

где f_t – забывающее состояние на текущем шаге t; W_f – весовая матрица для забывающего состояния; $[h_{t-1}, x_t]$ – конкатенированный вектор скрытого состояния на предыдущем шаге и текущего входа; b_f – вектор смещения для забывающего состояния.

Входное состояние (Input gate): управляет тем, какая информация будет добавлена в состояние памяти на текущем шаге. Сначала рассчитывается промежуточная величина $\widetilde{C}_t$, которая будет добавлена к состоянию памяти. Также используется сигмоидная функция для решения, какая информация из $\widetilde{C}_t$ будет использована:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i), \tag{6}$$

$$\widetilde{C}_t = \tanh(W_C * [h_{t-1}, x_t] + b_C), \tag{7}$$

где i_t – входное состояние, которое контролирует, сколько информации будет добавлено в память; $\widetilde{C}_t$ – кандидатное состояние памяти на шаге t; W_i, W_C – весовые матрицы для входного состояния и состояния памяти; b_i, b_C – векторы смещения.

Обновление состояния памяти (Cell state update): следующим шагом происходит обновление состояния памяти C_t. Сначала забывающее состояние f_t умножается на предыдущее состояние памяти C_{t-1} , что позволяет забыть часть старой информации. Затем добавляется новая информация, которая определяется входным состоянием и кандидатным состоянием [5]:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t, \tag{8}$$

где C_t – обновлённое состояние памяти на шаге t, f_t – забывающее состояние на текущем шаге t, i_t – входное состояние, $\widetilde{C}_t$ – кандидатное состояние памяти на шаге t.

Выходное состояние (Output gate): контролирует, какая информация из состояния памяти C_t будет передана на следующий шаг и на выход модели. Для этого используется сигмоидная функция и гиперболический тангенс для масштабирования значения:

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o), \tag{9}$$

$$h_t = o_t * \tanh(C_t), \tag{10}$$

где o_t – выходное состояние, которое управляет тем, какая информация будет передана дальше; h_t – скрытое состояние (выход из LSTM) на шаге t; $\tanh(C_t)$ – активация гиперболическим тангенсом из обновлённого состояния памяти.

Каждый шаг LSTM можно интерпретировать так:

- Забывающее состояние f_t – на этом шаге модель решает, какую часть информации из предыдущего состояния памяти нужно забыть. Например, если информация больше не актуальна, она будет игнорироваться.
- Входное состояние i_t – модель решает, какие новые данные стоит добавить в память. Для этого вычисляется $\widetilde{C}_t$, который представляет собой кандидатную информацию.
- Обновление памяти – новое состояние памяти C_t, формируется как комбинация старого состояния и новой информации. Это позволяет сохранять долговременные зависимости.
- Выходное состояние h_t – решает, какая информация из состояния памяти будет передана на следующий шаг [6].

3. Реализация прогноза с помощью SSA и LSTM

Перед началом реализации прогнозной функции необходимо собрать данные о котировках акций за определённый промежуток времени. Для этого необходимо обратиться к официальному API Московской биржи MoexAlgo.

Обратим внимание на рис. 1. Здесь создаётся объект sber класса Ticker, который представляет тикер акций компании (в данном случае компании «Сбер», тикер

SBER на бирже). Это объект, через который можно получать финансовые данные для указанной компании.

Создаётся переменная current_day, которая содержит текущую дату, но уменьшенную на один день. Т. е. это вчерашний день (если сегодня, например, 21 ноября 2024 года, то переменная current_day будет содержать «2024-11-20»).

Переменная first будет содержать дату, которая была 900 дней назад от текущего дня. Это может быть сделано для того, чтобы получить исторические данные о котировках за несколько лет (900 дней – это примерно 2,5 года).

```
yndx = Ticker('SBER')
current_day = (dt.datetime.now().date()- dt.timedelta(days=1)).strftime("%Y-%m-%d")
first = (dt.datetime.now().date()- dt.timedelta(days=900)).strftime("%Y-%m-%d")

df_full = yndx.candles(start=first, end=current_day, period='1d')
```

```
from sklearn import preprocessing
df = df_full.copy()
x = df[['close']] #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled)
```

Рис. 1. Сбор информации OHLC.

На основе объекта sber вызывается метод candles, который запрашивает данные о котировках акций компании за период с first (900 дней назад) по current_day (вчерашний день). Параметр period=’1d’ указывает, что данные должны быть в виде дневных свечей (т. е. информация о цене открытия Open, закрытия Close, максимуме High и минимуме Low за каждый день).

В результате выполнения этого кода переменная df будет содержать DataFrame с историческими данными о ценах акций компании «Сбер» (тикер SBER) за последние 900 дней. Эти данные можно использовать для анализа или построения графиков (см. рис. 2).

Следующим этапом будет формирование функции SSA. Функция ssa выполняет роль SSA на временном ряду series:

- Window – это размер окна, которое используется для разбиения временного ряда на части (траекторная матрица X).
- Создаётся матрица X, состоящая из окон размера window. Каждый столбец этой матрицы – это последовательность значений ряда, размером с окно.
- Затем выполняется SVD матрицы X на три матрицы: U, S, и VT. Эти матрицы представляют собой составляющие разложения (сингулярные векторы и значения).

На рис. 3 видно часть кода, где задаётся размер окна (window = 20) и выполняется SSA с использованием функции ssa. Результат разложения сохраняется в переменные U, S, и VT:

U – матрица левых сингулярных векторов.

S – диагональная матрица сингулярных значений.

	open	close	high	low	value	volume	begin	end
0	137.70	141.15	141.77	136.77	9.812403e+09	69963540.0	2022-06-27	2022-06-27 23:59:59
1	141.75	139.99	142.35	138.50	7.675925e+09	54863240.0	2022-06-28	2022-06-28 23:59:59
2	139.80	132.80	139.89	132.75	7.517498e+09	55164310.0	2022-06-29	2022-06-29 23:59:57
3	132.50	125.20	135.44	123.55	1.177445e+10	91499020.0	2022-06-30	2022-06-30 23:59:59
4	124.13	129.91	132.00	122.50	7.809272e+09	60628570.0	2022-07-01	2022-07-01 23:59:59
...	...	...	...	...	...	...	...	...
624	231.00	224.55	233.79	224.30	1.661712e+10	72417390.0	2024-12-04	2024-12-04 23:59:59
625	224.57	233.48	233.75	222.82	1.985159e+10	86989140.0	2024-12-05	2024-12-05 23:59:59
626	233.63	237.84	238.25	231.52	1.908810e+10	81096070.0	2024-12-06	2024-12-06 23:59:59
627	239.29	237.29	240.51	236.75	1.252639e+10	52493640.0	2024-12-09	2024-12-09 23:59:59
628	238.00	230.82	238.01	230.65	1.371039e+10	58740230.0	2024-12-10	2024-12-10 23:59:59

Рис. 2. Результат сбора информации о котировках акции Сбербанка

VT – матрица правых сингулярных векторов.

Восстановление тренда осуществляется путём вычисления произведения матриц U, S, и VT. Для тренда используется первая компонента (первый столбец из U и первая строка из VT), умноженная на соответствующие сингулярные значения из S. trend_component_mean содержит среднее значение каждого столбца тренда, что помогает сгладить и интерпретировать тренд.

Восстановление сезонного компонента производится аналогично тренду, но теперь используется вторая компонента (второй столбец из U и вторая строка из VT), умноженная на соответствующие сингулярные значения из S. Seasonal_component_mean содержит средние значения каждого столбца сезонного компонента.

Далее на рис. 4 выводятся графики, которые отображают исходный временной ряд, трендовую и сезонную составляющую.

В коде на рис. 5 выполняется подготовка модели для прогноза временных рядов с использованием LSTM (длительная кратковременная память).

Далее приведено пошаговое описание действий:

- create_lstm_features – функция, которая будет генерировать обучающие примеры для LSTM, где каждый пример – это окно из window предыдущих значений временного ряда.
- window=60 – размер окна для формирования признаков. Каждое окно будет содержать 60 предыдущих значений.
- Функция возвращает два массива: features (признаки для модели) и target (цели для предсказания, т. е. следующее значение ряда).
- MinMaxScaler масштабирует данные в диапазон (0, 1), что помогает модели лучше обучаться

```
# Преобразуем в DataFrame
df = pd.DataFrame({'time': df.index, 'value': series})

# Функция для выполнения SSA
def ssa(series, window):
    n = len(series)
    X = np.asarray([series[i:i + window] for i in range(n - window + 1)])
    U, S, VT = svd(X)
    return U, S, VT

# Выделение компонентов
window = 20  # Размер окна
U, S, VT = ssa(series, window)

# Восстановление тренда
trend_component = np.dot(U[:, 0].reshape(-1, 1), S[0] * VT[0, :].reshape(1, -1))
trend_component_mean = []
for j in range(0,len(trend_component)):
    trend_component_mean.append(trend_component[j].mean())

# Восстановление сезонного компонента
seasonal_component = np.dot(U[:, 1].reshape(-1, 1), S[1] * VT[1, :].reshape(1, -1))
seasonal_component_mean = []
for j in range(0,len(seasonal_component)):
    seasonal_component_mean.append(seasonal_component[j].mean())
```

Рис. 3. Код SSA

- Разделяем данные на тренировочную и тестовую выборки. Обычно для временных рядов используется 80 % данных для тренировки и 20 % для тестирования.
- train_size вычисляется как 80 % от общего количества примеров в данных.
- Первый слой LSTM содержит 100 единиц и параметр return_sequences=True, чтобы передавать выходные данные в следующий LSTM слой.
- Для предотвращения переобучения добавляется слой Dropout с вероятностью 0,2.
- Второй слой LSTM содержит 100 единиц и return_sequences=False, так как он является последним слоем.
- Выходной слой содержит 1 нейрон, который предсказывает одно значение.
- Компиляция модели используется оптимизатор adam и функция потерь mean_squared_error, подходящая для задачи регрессии.
- early_stopping добавляет механизм раннего завершения для предотвращения переобучения. Если валидационная ошибка не улучшится в течение 20 эпох, обучение будет остановлено, и веса модели будут восстановлены на лучшее состояние.

После построения функции обучения модели необходимо её запустить. Эта часть кода представлена на рис. 6.

Следующий этап – это запуск самого прогноза (рис. 7):

- model_lstm.predict(X_test) – делает прогнозы для тестовой выборки X_test. Это создаёт массив предсказанных значений, который сохраняется в переменной

Рис. 4. Графики SSA

y_pred_lstm.

- r2_score(y_test, y_pred_lstm) – функция из библиотеки sklearn, которая вычисляет R^2 между истинными значениями y_test и предсказанными значениями y_pred_lstm. Значение R^2 выводится на экран. Если оно близко к 1, это означает, что модель хорошо предсказывает данные.
- last_window = scaled_data[-60:].reshape(1, 60, 1) – создаётся окно из последних 60 значений, которые будут использоваться для прогноза. Оно преобразуется в нужный формат (3D: sample, time steps, features).
- В цикле на протяжении 10 итераций модель делает прогноз для следующего дня. На каждом шаге: y_forecast_lstm = model_lstm.predict(last_window) – модель прогнозирует следующее значение.
- last_window = np.append(last_window[:, 1:, :], y_forecast_lstm.reshape(1, 1, 1), axis=1) – для следующего шага прогнозирования окно обновляется. Из него «выбрасывается» первое значение, а предсказанное значение добавляется в конец. Это позволяет использовать новые предсказания как входные данные для следующих шагов прогноза.

Финальным этапом является представление прогноза на 10 дней на графике. Это можно увидеть на рис. 8.

После обучения модели LSTM для предсказания временного ряда результаты на тренировочных и тестовых данных продемонстрировали отличные показатели. Ко-

```
close_prices = pd.DataFrame(trend_component_mean).values
close_prices = close_prices.flatten()
# Преобразование временного ряда в структуру для XGBoost
def create_lstm_features(data, window=60):
    features, target = [], []
    for i in range(window, len(data)):
        window_data = data[i-window:i]
        features.append(window_data)
        target.append(data[i])
    return np.array(features), np.array(target)

# Нормализация данных для LSTM
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(close_prices.reshape(-1, 1))

# Создаем признаки и цели для LSTM
X_lstm, y_lstm = create_lstm_features(scaled_data, window=60)

# Разделение на тренировочную и тестовую выборки
train_size = int(len(X_lstm) * 0.8)
X_train, X_test = X_lstm[:train_size], X_lstm[train_size:]
y_train, y_test = y_lstm[:train_size], y_lstm[train_size:]

# Ресайзим данные для LSTM (3D: samples, time steps, features)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Создание модели LSTM с улучшениями
model_lstm = Sequential()

# Добавляем первый слой LSTM с Dropout для предотвращения переобучения
model_lstm.add(LSTM(units=100, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model_lstm.add(Dropout(0.2))  # Dropout для регуляризации

# Добавляем второй слой LSTM
model_lstm.add(LSTM(units=100, return_sequences=False))
model_lstm.add(Dropout(0.2))  # Dropout для регуляризации

# Выходной слой
model_lstm.add(Dense(units=1))

# Компиляция модели
model_lstm.compile(optimizer='adam', loss='mean_squared_error')

# Добавляем раннее завершение для предотвращения переобучения
early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)
```

Рис. 5. Построение нейронной сети

эффициент детерминации R^2 на тренировочных данных составил $0{,}9957$, что свидетельствует о высокой степени объясняемой модели изменчивости данных и отличной обучаемости. Модель прекрасно справляется с воспроизведением закономерностей в тренировочных данных.

На тестовых данных коэффициент $R^2 = 0{,}9815$, что также указывает на высокую способность модели обобщать на новые данные и точность её предсказаний. Это значение близко к 1, что подтверждает хорошее качество прогноза и уверенность модели в предсказаниях.

Средняя абсолютная ошибка (MAE) составила $0{,}0119$, что указывает на очень низкую среднюю ошибку предсказаний по сравнению с реальными значениями. Это означает, что модель делает точные прогнозы, отклоняясь от реальных данных всего на несколько тысячных единиц.

```
model_lstm.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=1, callbacks=[early_stoppi
Epoch 1/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 55ms/step - loss: 0.0013 - val_loss: 2.7688e-04
Epoch 2/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step - loss: 0.0012 - val_loss: 0.0011
Epoch 3/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - loss: 0.0014 - val_loss: 0.0012
Epoch 4/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - loss: 0.0011 - val_loss: 2.8362e-04
Epoch 5/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - loss: 0.0016 - val_loss: 2.0498e-04
Epoch 6/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 0.0013 - val_loss: 0.0025
Epoch 7/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - loss: 0.0014 - val_loss: 1.5655e-04
Epoch 8/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - loss: 9.4336e-04 - val_loss: 6.3167e-04
Epoch 9/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - loss: 0.0011 - val_loss: 0.0027
Epoch 10/100
```

Рис. 6. Обучение нейронной сети

```python
# Прогнозирование на тестовых данных
y_pred_lstm = model_lstm.predict(X_test)

# Вычисляем R^2 на тестовых данных
r2 = r2_score(y_test, y_pred_lstm)
print(f"R^2 на тестовых данных: {r2:.4f}")

# Прогнозирование на 10 дней вперед
last_window = scaled_data[-60:].reshape(1, 60, 1)  # последние 60 значений для прогноза
forecast_10_days_lstm = []
for _ in range(10):
    y_forecast_lstm = model_lstm.predict(last_window)
    forecast_10_days_lstm.append(y_forecast_lstm[0][0])
    last_window = np.append(last_window[:, 1:, :], y_forecast_lstm.reshape(1, 1, 1), axis=1)

# Обратная трансформация прогноза
forecast_10_days_lstm = scaler.inverse_transform(np.array(forecast_10_days_lstm).reshape(-1, 1))
```

Рис. 7. Реализация прогноза на 10 дней

Среднеквадратичная ошибка (MSE) оказалась ещё более низкой – $0{,}0002$, что ещё раз подтверждает высокую точность модели. Меньшие значения MSE означают, что модель минимизирует ошибки, особенно более значительные, которые могли бы существенно повлиять на прогноз.

В целом модель показывает отличные результаты как на тренировочных, так и на тестовых данных, и её можно считать высокоэффективной для решения задачи предсказания временного ряда. Модель может быть использована для дальнейших прогнозов, таких как предсказание на 10 дней вперёд, с уверенностью в точности предсказаний.

Литература

1. Голяндина Н.Э. Метод «Гусеница – SSA»: анализ временных рядов: учеб. пособие. СПб: С.-Петерб. гос. ун-т, 2004. 76‘с.

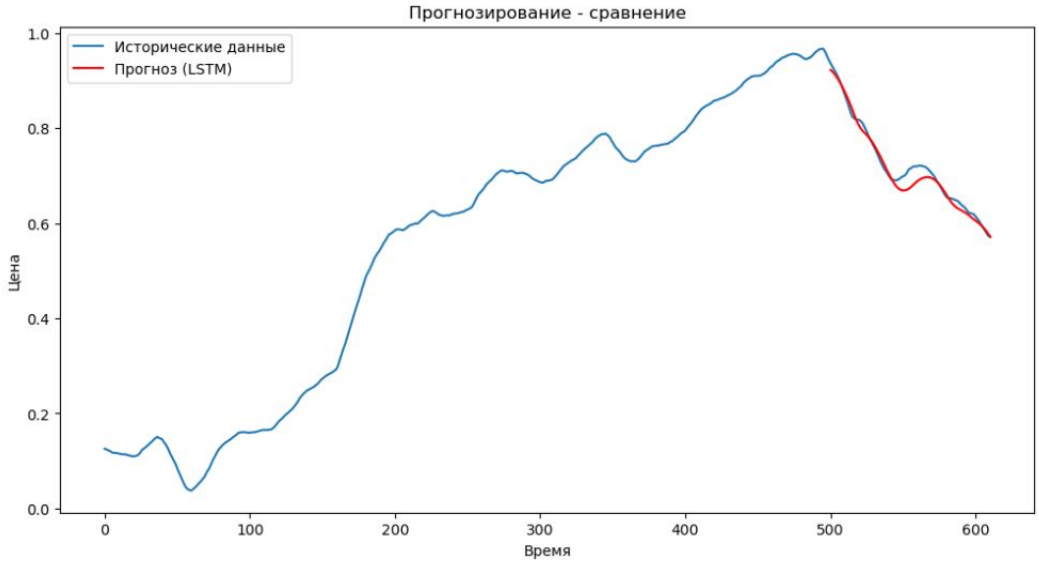

Рис. 8. Результат прогнозной функции

2. Главные компоненты временных рядов: «Гусеница-SSA» / под ред. Д. Л. Данилова, А.А. Жиглявского. СПб: Пресском, 1997.
3. Цыплаков А. Введение в прогнозирование в классических моделях временных рядов // Квантиль. 2006. № 1. С. 3–19.
4. Stratigakos A., Bachoumis A. Vita V., Zafiropoulos E. Short-Term Net Load Forecasting with Singular Spectrum Analysis and LSTM Neural Networks // Energies. 2021. Vol. 14. 10.3390/en14144107.
5. Duan J., Zuo H., Duan J., Chang M., Chen B. (2021). Short-term wind speed forecasting using recurrent neural networks with error correction // Energy. 2021. Vol. 217. 119397. 10.1016/j.energy.2020.119397.
6. Ye Zhang, Weide Li. SSA-LSTM neural network for hourly PM2.5 concentration prediction in Shenyang, China // J. Phys.: Conf. Ser. 2021. 1780 012015

TIME SERIES FORECASTING MODEL BASED ON SINGULAR SPECTRAL ANALYSIS AND LSTM NEURAL NETWORK

A.D. Baydalin
Graduate student, e-mail: alexander.baydalin@gmail.com
V.V. Varlamov
Dr.Sc. (Phys.-Math.), Professor, e-mail: varlamov@sibsiu.ru

Siberian State Industrial University, Novokuznetsk, Russia

Abstract. The article is devoted to the development of a model for predicting the company's share price using singular spectral analysis in combination with the LSTM neural network. The article suggests an approach to analyzing stock price time series based on the use of singular spectral analysis, which allows us to identify hidden patterns and trends in the data,

as well as eliminate noise, improving forecast accuracy. The technique involves decomposing a time series into several components, which contributes to a deeper understanding of price dynamics and increases the reliability of predictions. The experimental results showed the high efficiency of the proposed method in comparison with traditional statistical and machine analysis methods, which opens up new opportunities for optimizing investment strategies and analyzing financial markets.

Keywords: singular spectral analysis, time series, financial markets, data analysis, investment strategies, neural networks.

Дата поступления в редакцию: 24.12.2024