Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Казанский (Приволжский) федеральный университет»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИЧЕСКОЙ СТАТИСТИКИ

Направление подготовки: 01.03.02 — Прикладная математика и информатика
Профиль: Системное программирование

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ ПРИ ПРОГНОЗИРОВАНИИ ВРЕМЕННЫХ РЯДОВ

Обучающийся 4 курса
группы 09-612 — Александров Н. А.

Руководитель
канд. физ.-мат. наук, доцент — Кареев И. А.

Заведующий кафедрой математической статистики
канд. физ.-мат. наук, доцент — Симушкин С. В.

Казань – 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В анализе данных отдельным классом задач выделяют анализ временных рядов — числовых данных определённого процесса, собранных в последовательные моменты времени с равными промежутками. Наличие только самих данных и их индексации во времени даёт относительно малое количество информации в сравнении с другими классами задач анализа (например регрессионного). При этом задачи анализа временных рядов часто возникают в анализе сигналов датчиков и прогнозировании цен.

Несмотря на ограниченность данных, выявление стабильной временной структуры и закономерностей в поведении ряда способно дать достаточное количество информации для анализа и прогнозирования случайного процесса. Временной ряд может иметь характерный тренд, сезонность (или даже несколько), а также линейную зависимость от промежутка предыдущих значений. В подобных ситуациях существуют различные методы выделения существующих зависимостей.

В частности, задача может рассматриваться в многомерной вариации — когда несколько временных рядов, индексируемых общим множеством времени, связаны между друг другом попарно или в совокупности. В последнем случае, методы могут использоваться для каждого ряда в отдельности, или, при возможности модифицирования этих методов, использовать одновременно все данные целиком.

В данной работе на основе данных о продажах множества товаров в сети магазинов рассматривалось применение подобных методов в целях анализа и прогнозирования временных рядов.

Представлены ежедневные данные о продажах 50 наименований в сети из 10 магазинов в период с января 2013 года по декабрь 2018 года, доступные в завершённом соревновании **Store Item Demand Forecasting Challenge**, проходившем на Kaggle. Цель соревнования: с помощью имеющихся данных осуществить прогноз продаж по всем товарам в каждом из магазинов на 3 месяца вперёд. В ходе данной работы реализация методов проводится и оценивается в контексте решения аналогичной задачи. Листинг программы по пунктам представлен в приложении А.

1 Предварительный анализ

Перед тем как заниматься непосредственным построением модели, необходимо оценить структуру и внешний вид данных. Данные представлены как 913000 записей, содержащие **Дату, Номер магазина, Номер товара** и **Количество проданных единиц**. Пример изображён в таблице 1:

Таблица 1 – Пример представления данных

Index	Date	Store	Item	Sales
433020	2013-09-16	8	24	56
722747	2017-01-17	6	40	24
757614	2017-07-09	5	42	41
558817	2013-03-03	7	31	39
196461	2015-12-16	8	11	70

Введем обозначение: $TS_{i,s} = TS(i, s, date)$ - временной ряд, определяющий продажи i-ого товара, в s-ом магазине в указанный день. Таким образом данные можно интерпретировать как 500 различных временных рядов $TS_{i,s}$, внешний вид которых, как можно увидеть на рисунке 1, схож между друг другом.

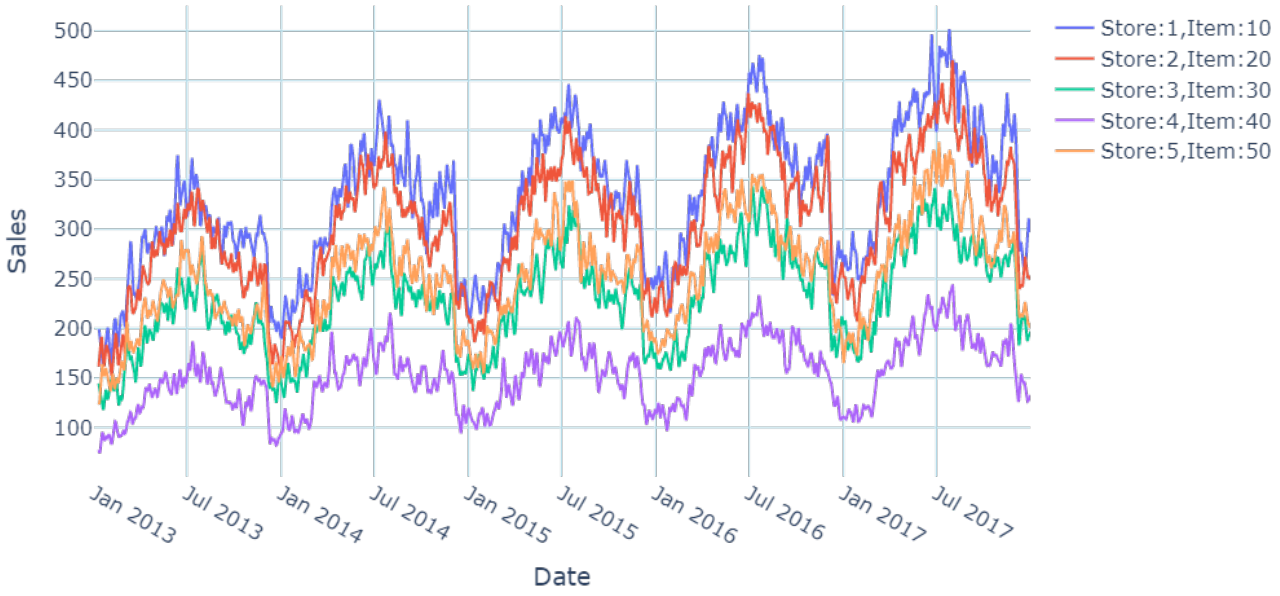


Рисунок 1 – Временные ряды $TS_{10,1}$,$TS_{20,2}$,$TS_{30,3}$,$TS_{40,4}$,$TS_{50,5}$

1.1 Сезонность, тренд

Из внешнего вида $TS_{i,s}$ можно утверждать о том, что, несмотря на различия в уровнях продаж, у каждого временного ряда есть восходящий тренд и сезонность. Визуально по графику отдельного ряда сезонность можно оценить годовым периодом. Точную оценку можно получить с помощью периодограммы.

Идея периодограммы возникает из спектрального анализа временных рядов и подробно рассматривается в материалах [1]. Временной ряд может рассматриваться, как сумма косинусоид с разными амплитудами и частотами. Пусть имеются данные за n временных точек. Задача: выявить значительные частоты в данных. Рассматривается множество гармоник — частот вида $\omega_j = j/n$, где $j = 1, 2, \ldots, n/2$.

Тогда временной ряд y_t можно представить в виде:

$$y_t = \sum_{j=1}^{n/2} \left[A\cos(2\pi\omega_j t + \phi) \right]. \tag{1}$$

Известно тригонометрическое преобразование:

$$A\cos(2\pi\omega t + \phi) = A\cos(\phi)\cos(2\pi\omega t) - A\sin(\phi)\sin(2\pi\omega t) \tag{2}$$

Пусть $\beta_1 = A\cos(\phi)$, $\beta_2 = -A\sin(\phi)$. Используя (2), выражение (1) принимает вид:

$$y_t = \sum_{j=1}^{n/2} \left[\beta_1 \left(\frac{j}{n}\right) \cos(2\pi\omega_j t) + \beta_2 \left(\frac{j}{n}\right) \sin(2\pi\omega_j t) \right] \tag{3}$$

Необходимо найти n параметров $\beta_1(j/n), \beta_2(j/n)$, для $j = 1, 2, \ldots, n/2$. Они могут быть получены приближением, как параметры регрессии. Другим, более эффективным способом, является метод быстрого преобразования Фурье. Данный алгоритм, использованный для вычислений, описан в материале [2].

После получения параметров $\beta_1(j/n), \beta_2(j/n)$, значения периодограммы вычисляются по формуле:

$$P\left(\frac{j}{n}\right) = \widehat{\beta}_1^2\left(\frac{j}{n}\right) + \widehat{\beta}_2^2\left(\frac{j}{n}\right) \tag{4}$$

Определенным выше методом строится периодограмма, вид которой изображен на рисунке 2. Полученные значения интерпретируются таким образом - относительно большие $P\left(\frac{j}{n}\right)$ говорят о важности частоты $\frac{j}{n}$ в объяснений осцилляций наблюдаемого ряда.

На основании данной периодограммы можно утверждать о наибольшей значимости частот 5 и 261, соответствующих годовому и недельному периоду соответственно. Данные свойства характерны для каждого $TS_{i,s} = TS(i, s, date)$

Визуально заметно наличие тренда у всех временных рядов. Рассмотрим относи-

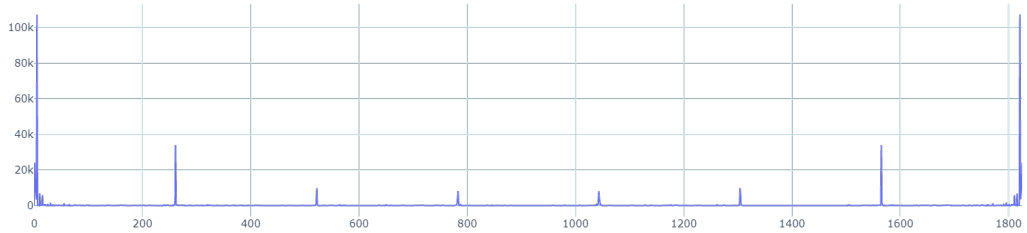


Рисунок 2 – Периодограмма для продаж 10-ого товара в 1-ом магазине

тельные изменения среднегодовых продаж $average_{i,s,year}\,TS$ в сравнении с глобальным средним $average_{total}\,TS$ (рисунок 3):

$$trend(year) = \frac{average_{i,s,year}\,TS}{average_{total}\,TS}. \tag{5}$$

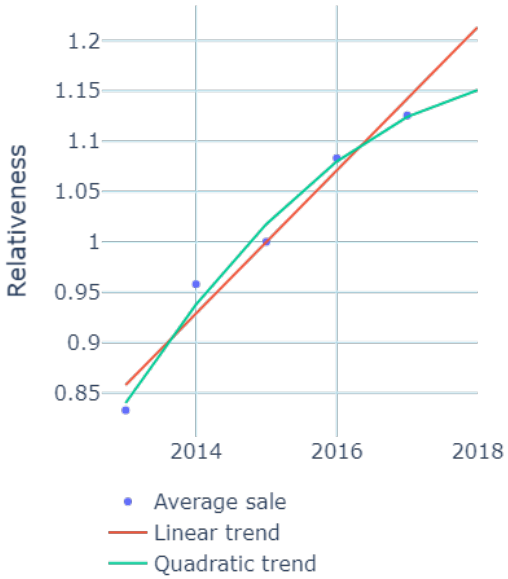


Рисунок 3 – Тренд.

Среднеквадратичное отклонение для квадратичного тренда меньше, чем для линейного (1.15 к 1.21), что может быть подозрением о наличии эффекта затухания - рост не может быть бесконечным и со временем скорость увеличения падает. Однако, эта разница слишком маленькая чтобы утверждать с высокой долей уверенности.

1.2 Выявление закономерностей, сводные таблицы

Наличие тренда и сезонностей - общая черта всех исследуемых рядов, однако уровень значений в зависимости от товара и магазина может меняться: один и тот же товар

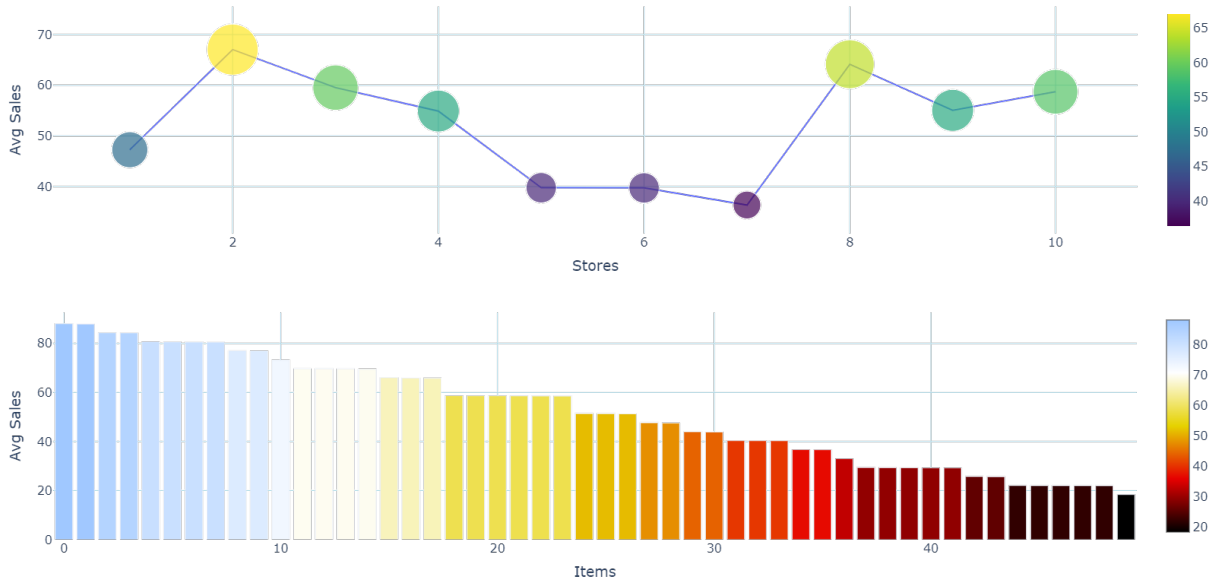
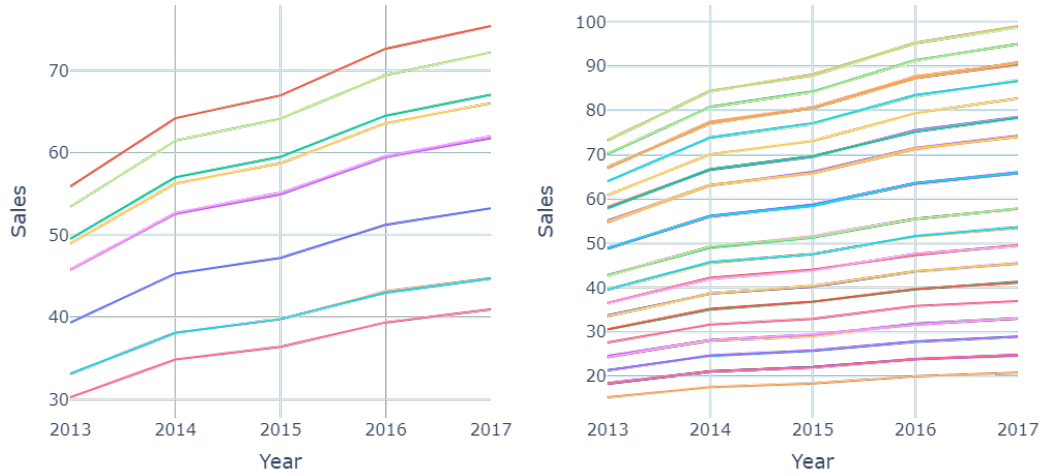


Рисунок 4 – Среднее значение продаж по каждому магазину(a) и по каждому товару(b)

может с различным успехом реализоваться в различных магазинах. Составив сводную таблицу, становится возможным визуально оценить средние продажи по каждому магазину и товару (рисунок 4).

Из сводной таблицы видно, что распределения уровня продаж различаются в зависимости от магазина и вида товара. Стоит заметить, что средний уровень продаж по магазинам имеет подозрительно ровные ступеньки - некоторые товары имеют почти одинаковые (с точностью до первого знака после запятой) средние уровни продаж по всем магазинам. Это повод заподозрить синтетическую природу данных.

Также могут быть рассмотрены среднегодовые данные продаж по каждому товару и магазину. Данные сводных таблиц представлены на рисунке 5. На графике 5 (b) видно, что те товары, что находились на одной ступени на графике 4 (b) не только имеют одинаковые среднее значение продаж за все 5 лет, но и их ежегодные средние значения также почти полностью совпадают. Более того, поведение роста среднего значения по всем товарам (и по всем магазинам) выглядит идентично.



(a) (b)

Рисунок 5 – Среднегодовые продажи по каждому магазину (a) и по каждому товару (b)

Данные выводы приводят к интересу вида относительного прироста среднегодовых продаж к среднему значению продаж (соответствующего магазина/товара). Результат представлен на рисунке 6. Относительный прирост цены за исследуемый период, независимо от того о каком товаре или магазине идёт речь, всегда один и тот же и совпадает с приростом для усреднённых данных (рисунок 3).

По полученным результатам закономерно предположить, что данные с высокой вероятностью синтетические, то есть получены искусственно. Для оценки данного утверждения рассмотрим по аналогии с ежегодными, относительные изменения по месяцам и дням недели. Полученные результаты представлены на рисунке 7. По графикам видно, что для каждого дня недели, для каждого месяца есть свои постоянные коэффициенты. В частности, эти постоянные для каждого месяца и дня недели очень хорошо объясняют выявленные периодограммой годовую и недельную сезонность:

$$season_{weekly}(weekday) = \frac{average_{i,s,weekday}\, TS}{average_{total}\, TS}, \tag{6}$$

$$season_{monthly}(month) = \frac{average_{i,s,month}\, TS}{average_{total}\, TS}, \tag{7}$$

1.3 Аналитическая модель, метрика оценки прогноза

Воспользовавшись равенствами 5, 6, 7, опишем модель вида:

$$TS(i, s, date) = TS_{base}(date) \cdot c(i, s) \cdot \eta_1(i, s, date) + \eta_2(i, s, date), \tag{8}$$

где η_1, η_2 — белый шум, а $c(i, s)$ и $TS_{base}(date)$ определяются формулами:

(a) (b)

Рисунок 6 – Относительные среднегодовые продажи по каждому магазину (a) и по каждому товару (b)

$$c(i,s) = average_{date}\, TS(i,s,date), \tag{9}$$

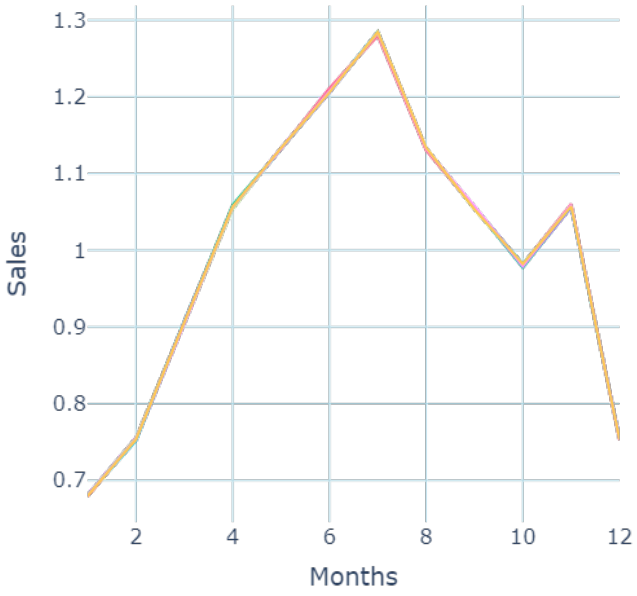
$$TS_{base}(date) = trend(year) \cdot season_{weekly}(weekday) \cdot season_{yearly}(month). \tag{10}$$

Оценка эффективности полученной выше и описываемых далее моделей будет определяться согласно метрике, использованной в ходе соревнования, откуда были взяты данные. Этой метрикой является SMAPE — симметричная средняя абсолютная процентная ошибка между прогнозом $\hat{y}_i$ и реальными значениями y_i:

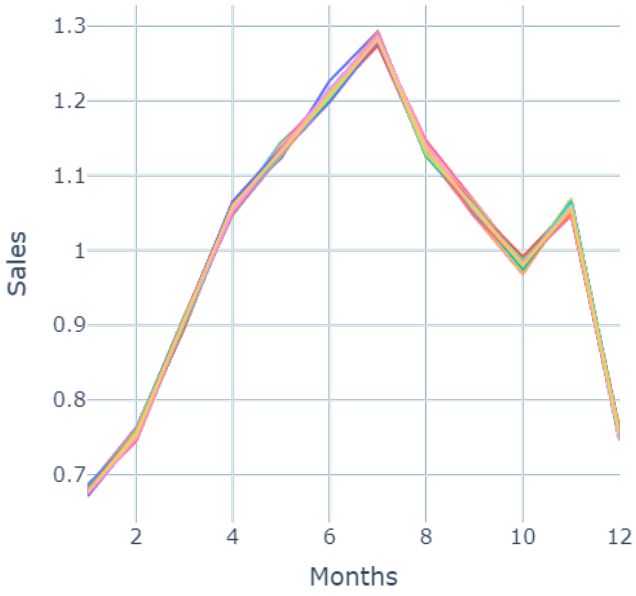
$$SMAPE = average_i \frac{|y_i - \hat{y}_i|}{\frac{1}{2}(y_i + \hat{y}_i)} \cdot 100\% \tag{11}$$

В ходе соревнования лучших результатов достигла подобная аналитическая модель. С помощью небольших модификаций аналитической модели (8) (суть которых не является целью данной работы) наилучшим решением, достигнутым исследователями, является SMAPE $= 12.58\%$.

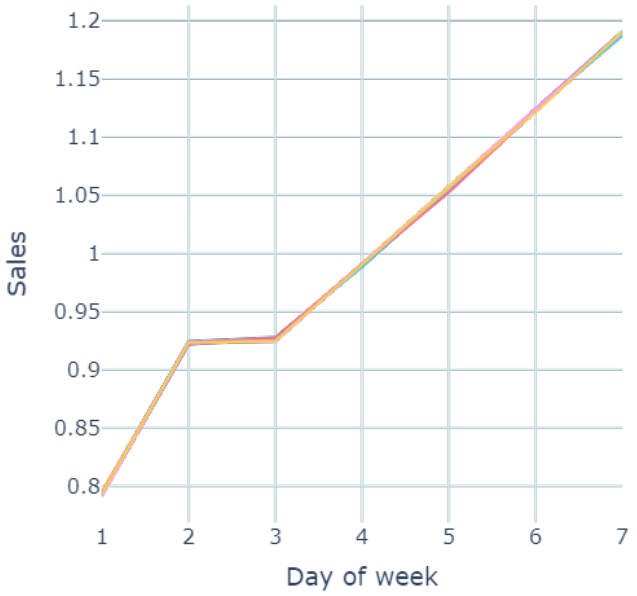
Предварительный анализ для задачи позволил описать эффективный способ ее решения. В реальных условиях данные редко обладают подобной регулярностью и явной структурой. В таких случаях для работы с временными рядами существуют различные универсальные методы и модели, способные уменьшить степень неопределённости о процессе.



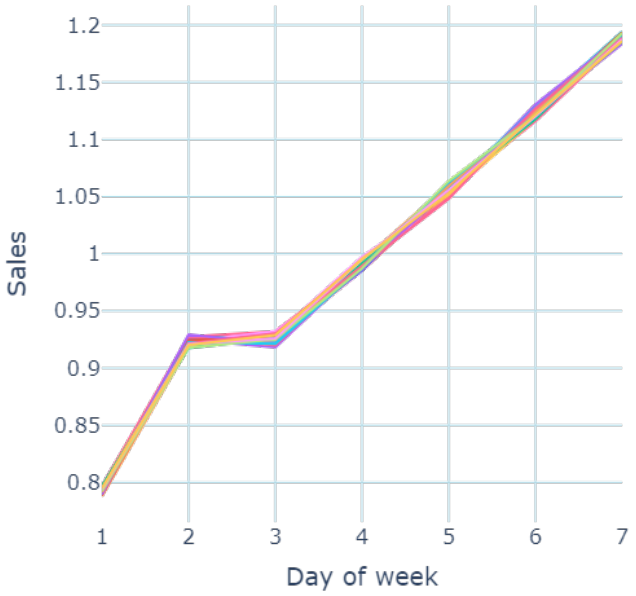
(a)



(b)



(c)



(d)

Рисунок 7 – Относительные средние продажи за месяц по каждому магазину (a) и по каждому товару (b). Относительные средние продажи за день недели по каждому магазину (c) и по каждому товару (d)

2 Статистические методы

Анализ временных рядов может осуществляться различными математико-статистическими методами. Некоторые из-за простой структуры могут быть малоэффективны. С другой стороны, простота моделей обеспечивает высокий уровень интерпретации, что может помочь в дальнейшем анализе. На начальном этапе рассматриваются способы анализа одномерных временных рядов. Итоговый результат получался объединением прогнозов сделанных для всех $TS_{i,y}$ по отдельности.

2.1 Линейная регрессия

Линейная регрессия наиболее простая в интерпретации и построении модель. Рассматривается модель вида:

$$x_t = \beta_0 + \beta_1 t + \varepsilon_t, \tag{12}$$

Подобная модель в анализе временных рядов может быть использована для приближения значения компоненты тренда, как линейного, так и степенного (на рисунке 8 представлен линейный тренд для $TS_{1,1}$).

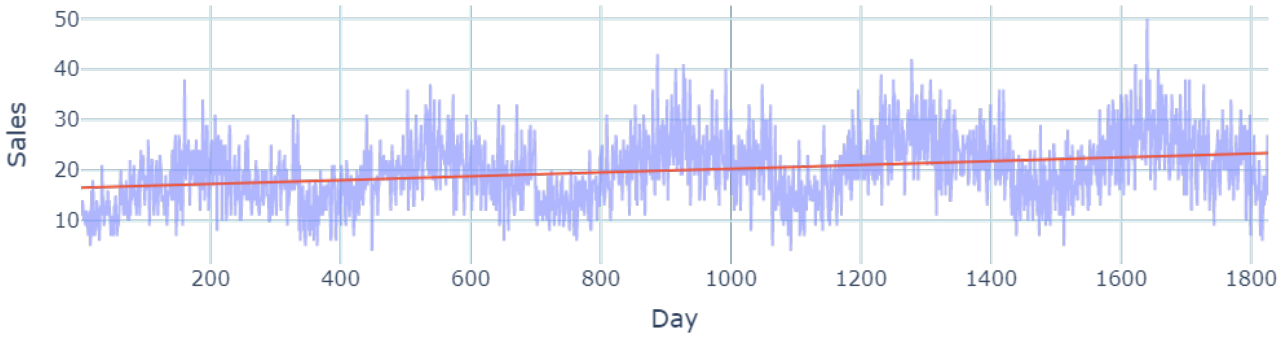


Рисунок 8 – — Линейный тренд на основе линейной регрессии

В практическом анализе для прогнозирования временного ряда регрессия оказывается неэффективной из-за своей простоты. А более сложная регрессия от нескольких предикторов требует подходящих многомерных данных. Как уже показал первичный анализ - значения временного ряда сильно зависят от таких признаков как **Наименование товара, Магазин, Месяц, День недели, Год, Тренд**. Преобразование записей таблицы 1 для явного выделения этих признаков отображено в таблице 2:

Таким образом была получена возможность воспользоваться многомерной линейной регрессией (13) для $TS_{i,s}$, учитывающей особенности сезонности, тренда, товара и места реализации путём добавления отвечающих за это предикторов.

Таблица 2 – Конструирование признаков из поля **Дата**

Index	Store	Item	Day,total	Day,week	Month	Year	Sales
	$x_{1,t}$	$x_{2,t}$	$x_{3,t}$	$x_{4,t}$	$x_{5,t}$	$x_{6,t}$	y_t
433020	8	24	259	0	9	2013	56
722747	6	40	1478	1	1	2017	24
757614	5	42	1651	6	7	2017	41
558817	7	31	62	6	3	2013	39
196461	8	11	1080	2	12	2015	70

$$y_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + +\beta_3 x_{3,t} + \beta_4 x_{4,t} + \beta_5 x_{5,t} + \beta_6 x_{6,t} + \varepsilon_t, \tag{13}$$

Применив данную модель ко всем временным рядам по отдельности с использованием кросс-валидации для общего прогноза будет получено значение SMAPE = 21.47% как для L1, так и для L2 регуляризаций. Данное значение чуть лучше чем то, которое можно получить, если в качества прогноза на определенный день вычислять среднее значение в этот день, за прошедшие 5 лет (SMAPE = 21.96%).

В ходе предварительного анализа была определена модель (8), из которой ясно видно, что поведение данных может описываться не аддитивным влиянием признаков, а мультипликативным. В таком случае, эффективность линейной модели будет заведомо ниже. Для того, чтобы улучшить ее качество, можно попробовать использовать полиномиальные: $x_{1,t}$, $x_{1,t}^2$, $x_{1,t}^3$, $x_{1,t} \cdot x_{2,t}$, $x_{1,t} \cdot x_{2,t}^2$, $x_{1,t} \cdot x_{2,t} \cdot x_{3,t}$, $\cdots$, $x_{6,t}^3$. С помощью полиномиальных признаков до 3 степени включительно, модель (13) можно модифицировать в модель вида:

$$y_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{1,t}^2 + \cdots + \beta_5 x_{5,t} x_{6,t}^2 + \beta_{84} x_{6,t}^3 + \varepsilon_t, \tag{14}$$

Для модели (14) показатель SMAPE снизился до 13.47% для L1-регуляризации (график 9) и до 14% для L2-регуляризации. Данный результат очень близок к тому, что был получен для аналитической модели (8), что показывает пользу тщательного конструирования признаков, способного обеспечить высокую точность прогноза для подобной базовой модели.

2.2 Экспоненциальное сглаживание

Ранее, в конце пункта 2.1.1, для тренда $average_{i,s,year}$ построение определялось не с

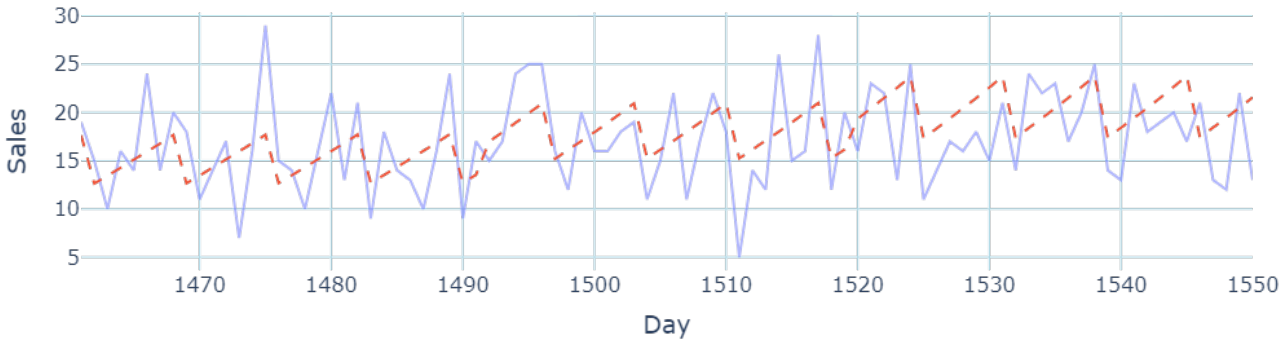


Рисунок 9 – Прогноз модели с полиномиальными признаками и L1-регуляризацией на валидационных данных

помощью линейной регрессии, а по формуле скользящего среднего:

$$Trend_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j}, \tag{15}$$

где $m = 2k+1$ определяет порядок скользящего среднего. Для среднегодового тренда было использовано $m = 365$. Такое большое значение m объясняется тем, что у данных есть годовой период, который вместе с трендом влияет на данные. Можно предположить, что, взяв среднее по периоду, получатся значения, на которые эффект сезонности уже не влияет. Модификацией скользящего среднего является взвешенная средняя, внутри которой наблюдениям придаются различные веса, в сумме дающие единицу:

$$\hat{y}_t = \sum_{n=1}^{k} \omega_n y_{t+1-n} \tag{16}$$

Если вместо последовательности значений взвешивать все доступные наблюдения, при этом экспоненциально уменьшая веса по мере углубления в исторические данные, полученная модель будет являть собой следующий инструмент для анализа временных рядов - экспоненциальное сглаживание, модель Брауна:

$$\hat{y}_t = \alpha \cdot y_t + (1 - \alpha) \cdot \hat{y}_{t-1} \tag{17}$$

Данная модель, на практике несет в себе малую информативность и может только наглядно сгладить ряд (рисунок 10). Это может помочь визуально оценить поведение зашумленных данных, однако в данной работе, интерес уделен определяемым ниже модификациям.

Для начала разобьём предполагается, что $TS_{i,s}$ может быть разбит на две составляющие — уровень l и тренд b. Для построения обоих компонент будет использоваться

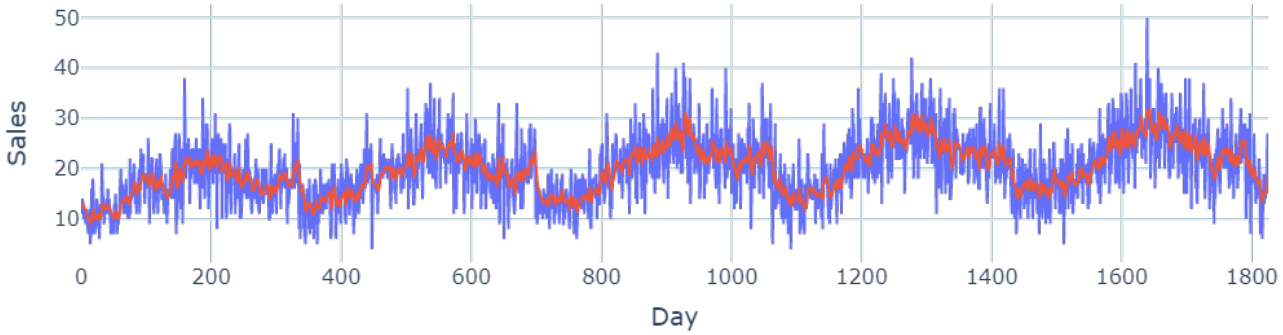


Рисунок 10 – Экспоненциальное сглаживание для $TS_{1,1}$

экспоненциальное сглаживание:

$$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}) \tag{18}$$

$$b_x = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \tag{19}$$

$$\hat{y}_{t+1} = \ell_t + b_t \tag{20}$$

Данный набор функций определяет двойное экспоненциальное сглаживание, модель Хольта, учитывающее тренд и уровень данных. Дальнейшим шагом является построение усложненной модели, учитывающей в дополнении к этому сезонность. На основе модели Хольта, Хольтом и Винтерсом была предложена модель, представленная в [3], раздел 6.4. Она имеет вид:

$$\ell_t = \alpha(y_t - s_{t-L}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \tag{21}$$

$$b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \tag{22}$$

$$s_t = \gamma(y_x - \ell_x) + (1-\gamma)s_{x-L} \tag{23}$$

$$\hat{y}_{t+m} = \ell_t + mb_t + s_{t-L+1+(m-1)modL}, \tag{24}$$

где m — шаг прогноза, L — длина периода сезона.

Метод (24), определяемый как тройное экспоненциальное сглаживание или модель Хольта-Винтерса с аддитивными трендом и сезонностью, сглаживает все сезонные компоненты через периоды. Для недельной сезонности, например, компоненты понедельника будут сглаживаться только с понедельниками. Для использования модели необходимо начальные инициализации тренда и сезонности, которые могут быть получены таким образом:

$$b = \frac{1}{L}\left(\frac{y_{L+1} - y_1}{L} + \frac{y_{L+2} - y_2}{L} + \cdots + \frac{y_{L+L} - y_L}{L}\right), \tag{25}$$

$$s_j = \frac{\left(\frac{y_j}{A_1} + \frac{y_{j+L}}{A_2} + \cdots + \frac{y_{j+\omega \cdot L}}{A_\omega}\right)}{\omega}, \tag{26}$$

где ω — частота периода за исследуемый временной промежуток, j — номер сезонной компоненты в периоде (для недельной это 1 — для понедельника, 2 — для вторника и т.д.). Из способа начальной инициализации следует, что условием метода (24) является такая длина ряда n, что $n \geq 2L$ — достаточное для получения первичного значения компоненты тренда.

Проблемы у метода возникают при учёте мультисезонности — наличия двух и более сезонов, с различными периодами. Стандартный подход метода в данном случае рассматривать наличие двух сезонов только как один сезон, длина периода которого $L = L_1 \cdot L_2$, где L_1 и L_2 — длины периодов сезонов. В нашем случае это равно $2 \cdot 365 \cdot 7 = 5110$ при $n = 7$. Однако требование начальной инициализации тренда для каждого отдельного ряда не может быть выполнено, ибо $n = 1826$ у каждого из $TS_{i,s}$.

Несмотря на данную проблему, модель (24) способна построить прогнозы, если брать в расчет только недельную или годовую сезонность, рисунок 11. В таком случае, использовав модель на каждому ряду по отдельности, SMAPE будет принимать значение 20.75% для недельной сезонности и 24.17% для годовой.

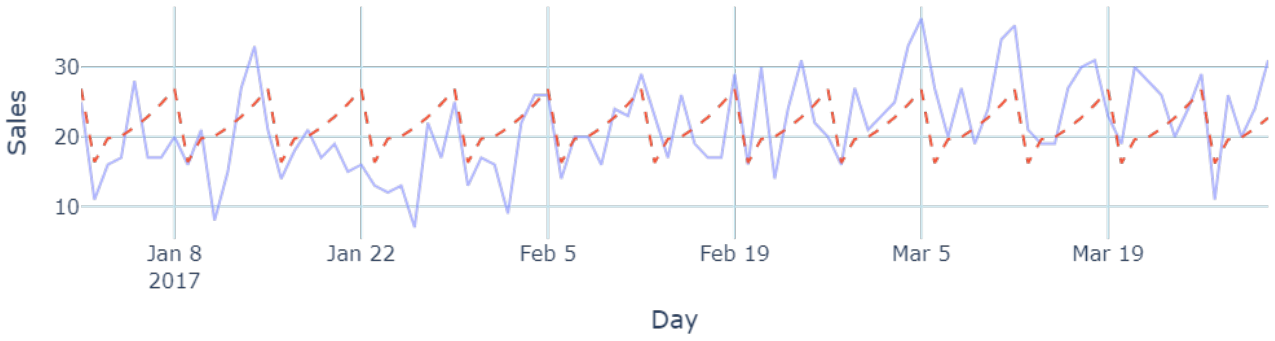


Рисунок 11 – Тройное экспоненциальное сглаживание с недельным сезоном для $TS_{1,10}$ на валидационной выборке

Игнорирование одного из сезонов приводит к потере информации о данных, что уменьшает привлекательность модели для прогнозирования, однако структура модели обладает высокой интерпретируемостью и коротким временем построения. Модели экспоненциального сглаживания являются представителем класса моделей пространственных состояний, определяемые наличием наблюдаемых (y_t) и ненаблюдаемых состояний процесса (b_t, s_t).

2.3 Декомпозиция, ARIMA

Для моделей (8) и (24) были предприняты попытки явным образом выявить закономерность роста данных от сезонных особенностей и тренда. Подобная идея может быть

обобщена представлением временного ряда в виде:

$$y_t = S_t + T_t + R_t \text{ для аддитивной модели,} \tag{27}$$

$$y_t = S_t \times T_t \times R_t \text{ для мультипликативной модели,} \tag{28}$$

где S_t — сезонная компонента, T_t — компонента тренда, R_t — компонента остатка.

Способов построения данных компонент существует несколько. Простейшим из них является алгоритм на основе скользящего среднего, описанный в [4], раздел 6.3:

1. Пусть сезонность определяется трендом с периодом m. Для построения $\hat{T}_t$ при четном m построение скользящего среднего осуществляется с использованием окна размера $2m$, для нечетного m - окно размером m.

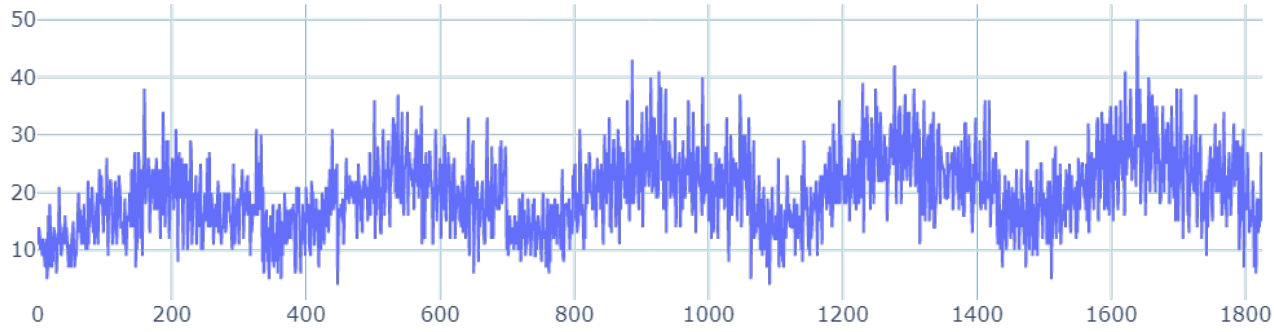
2. Для аддитивной модели полученные значения $\hat{T}_t$ вычитаются из y_t, для получения детрендированного ряда. Для мультипликативной детрендированный ряд получается как $y_t/\hat{T}_t$

3. Используя детрендированный ряд сезонная компонента $\hat{S}_t$ строится из средних для каждого момента периода по всем данным. То есть для годовой сезонности это среднее по всем 1 января, имеющимся в данных, по 2 января и т.д.

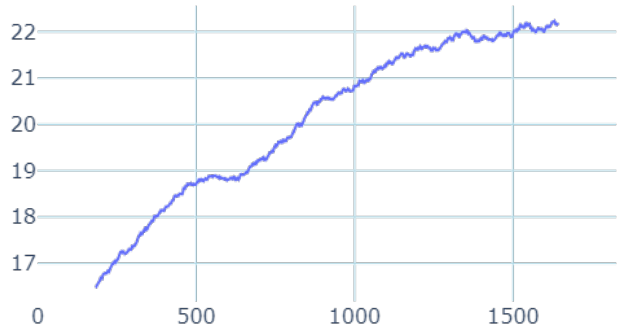
4. Остаток $\hat{R}_t$ вычисляется с помощью полученных компонент $\hat{T}_t$ и $\hat{S}_t$. Для аддитивной модели: $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$; для мультипликативной: $\hat{R}_t = \frac{y_t}{\hat{T}_t \cdot \hat{S}_t}$

Более сложные модели могут определять в качестве тренда ранее упомянутую регрессию, а для сезонной компоненты - вычисленные на основе периодограммы значения для самых сильных частот (предложено в [5], раздел 3).

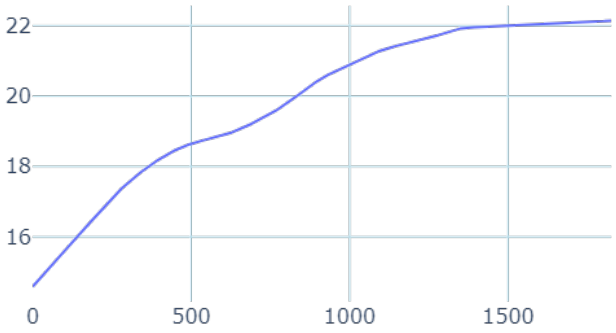
Одним из наиболее эффективных методов декомпозиции считается STL-декомпозиция, построенная на использовании локальной регрессии, описанной в работе [6]. Примеры декомпозиции с использованием различных методов отображены на графике (12).



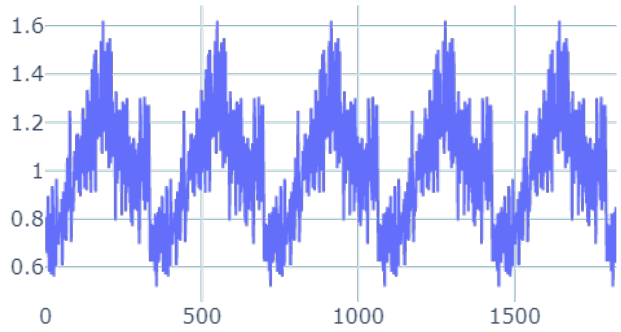
(a)



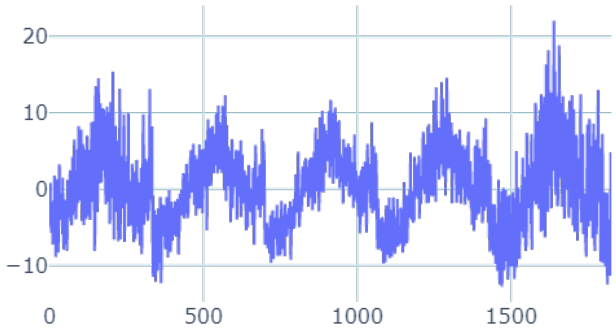
(b)



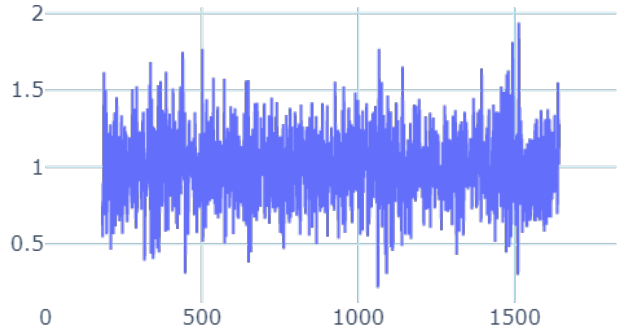
(c)



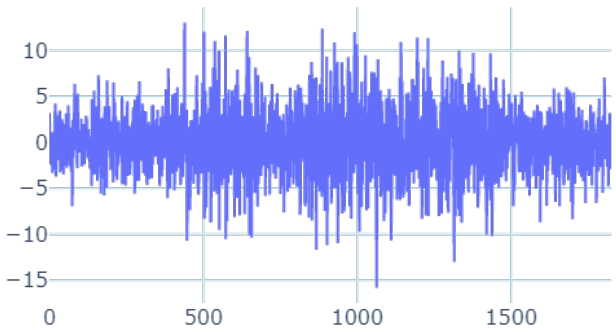
(d)



(e)



(f)



(g)

Рисунок 12 – Декомпозиция $TS_{i,s}$: (a) - рассматриваемый ряд, (b), (d), (f) - декомпозиция на основе скользящего среднего, (c), (e), (g) - STL-декомпозиция

Полученный компонент R_t может обладать временными особенностями, которые часто описываются моделями ARIMA (p, d, q), авторегрессивной моделью скользящего среднего, имеющей вид:

$$x'_t = \phi_1 x'_{t-1} + \cdots + \phi_p x'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t, \tag{29}$$

где x'_t — продифференцированный d-раз временной ряд $y'_t = y_t - y_{t-1}$, ε_{t-j} — ошибки j-ого прогноза, ε_t — белый шум. Порядок модели p и q определяется из графиков автокорреляций и частичных автокорреляций, а d из необходимости стационарности используемого моделью временного ряда.

Прогноз ряда $TS_{i,s}$ был разбит на три подзадачи - прогнозирование T_t, S_t, R_t, где каждую компоненту можно строить различными методами. В качестве примера, самый простой метод — декомпозиция тренда и сезонности на основе скользящего среднего, с последующим анализом остатка как ARIMA процесса.

Декомпозиция с годовой сезонностью для рассматриваемых одномерных временных рядов определяет остаток, анализ автокорреляций и частичных автокорреляций (рисунок 13) которого показывает наличие значимых корреляций данных с недельным периодом. Это говорит о наличии второй - недельной сезонности, что уже было подтверждено определёнными выше моделями.

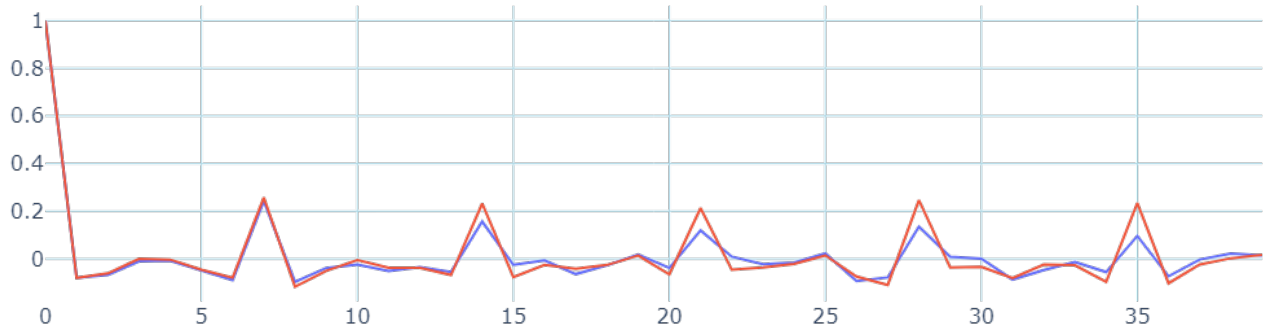


Рисунок 13 – Графики частичной автокорреляции (синий) и автокорреляции (красный)

После выделения новой сезонности из остатка $\hat{R}_t$, возникает новый остаток $R_t^{(2)}$, для которого можно опять оценить ARIMA модель. Но уместно задаться вопросом: сколько за эти два последовательных преобразования исходного ряда было извлечено реальной информации о структуре данных, а сколько извлечено из зашумления. Каждое преобразование данных приводит к увеличению влияния шума на информативность выделяемых особенностей. Чтобы избежать повторных преобразований, можно попробовать использовать линейную регрессию с ARIMA ошибкой, относительно исходного временного ряда.

Взяв в качестве основы способ представления сезонности через ряды Фурье, вре-

менной ряд можно представить как регрессию вида:

$$y_t = T_t + \sum_{j=1}^{M} \sum_{k=1}^{K_j} [\alpha sin(\frac{2\pi kt}{p_k}) + \beta cos(\frac{2\pi kt}{p_j})] + N_t, \tag{30}$$

где M - количество сезонностей, K_j - количество рядов Фурье для j-ого сезона, определяемое исходя из стремления минимизации информационного критерия Акаике для результата, p_j - период j-ого сезона, N_t - ARIMA процесс.

Линейные регрессии с ARIMA ошибкой можно строить другими способами, которые реализованы для большинства языков программирования работающих со статистикой. Чаще других в таких задачах встречаются методы работающие с максимальным правдоподобием, также рассматриваемыми в [1], или с упомянутыми выше пространственными состояниями, более подробно рассмотренными в [8].

Результаты вычислений показали SMAPE = 13.46% для модели (30) с двумя сезонами и ARIMA (2,0,1). Для сравнения модель использующая только недельную сезонность с помощью ARIMA (7,0,7) получила результат в 17.68%. Однако для вычисления значений для модели (30) для всех $TS_{i,s}$ потребовалось очень много времени в сравнении со всеми использованными выше значениями (в 7 раз больше чем модели (14), в 25 раз больше модели (24) и более чем в 200 раз больше модели (8)).

2.4 Многомерный случай

Рассмотренные выше методы были использованы на всех временных рядах $TS_{i,s}$ в отдельности, а затем прогнозы этих моделей были объединены в итоговый прогноз. Продолжением исследования стала попытка использовать модель для анализа, задействующая все данные.

- В случае линейной регрессии использование всех данных рассматривалось путем добавление в модель (13) двух дополнительных предикторов, отвечающих за номер магазина и товара, и соответствующее увеличение количества признаков для полиномиальной модели (14). Обучение на полном объёме данных привело к значениям SMAPE 45.38% и 40.18% для обычной модели и модели с полиномиальными признаками соответственно, обе модели использовали L1-регуляризацию.

- В теории временных рядов рассматриваются векторные VAR и VARMA модели, описание использования которых можно найти в [7]. Они описывают векторные временные ряды, компоненты которых зависят от всех рядов согласно модели (30) при отсутствии дифференцирования. Наименьшая по количеству параметров из них - VAR(1). Для ее обучения для 500 рядов требуется 25000 параметров для оценки. В таком случае встаёт вопрос вычислительных стоимостей и погрешности. При этом

польза данного подхода ставится под сомнение: использование VARMA с различными параметрами на малом объеме данных (5 наборов $TS_{i,j}$ с общими i или j) показало результаты в среднем хуже, чем комбинация прогнозов моделями (30) для рядов этого набора по отдельности.

Для модели Хольта-Винтерса, (24), возможности использовать многомерный подход найдено не было.

3 Методы машинного обучения

3.1 Предобработка временных рядов для обучения нейронных сетей

Анализ временных рядов нейронными сетями в работе рассматривался двумя способами.

Первый способ предполагает наличие у временного ряда сильной временной зависимости, которая проходит вдоль ряда и может быть определена по тому как меняются промежутки данных в течении движения "окна"по данным.

Такая модель требует преобразования исходных данных таким образом, чтобы одновременно на вход нейросети подавалось не одно значение y_t, а целый промежуток, выходным значением которого будет последующий промежуток значений (схема на рисунке 14), который может быть непосредственно за первым или несколько дальше. Архитектуры в таком случае могут представлять собой как и простой многослойный персептрон MLP, так и более сложные рекуррентные и нейронные сети LSTM, CNN. Во всех этих случаях стандартное представление данных строится с помощью скользящего окна.

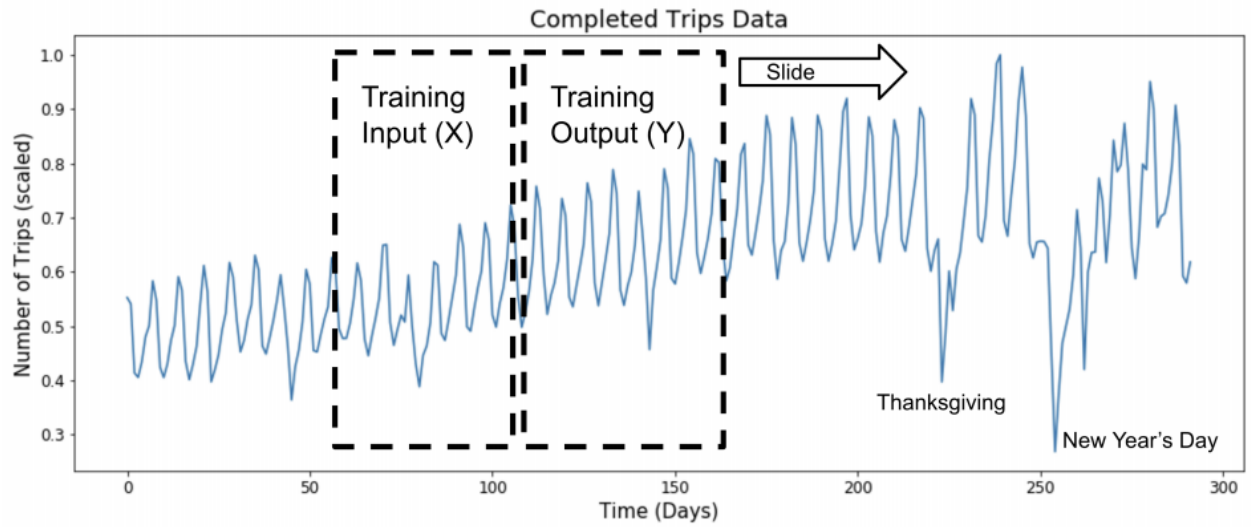


Рисунок 14 – Представление входных данных и целей скользящим окном

Второй подход использует embedding-слой, который преобразует временную индексацию категориальными признаками (**месяцем, днем недели, днем месяца**) момента y_t в вектор с непрерывными вещественными значениями, а затем использует этот вектор для обучения.

Первый метод более универсален - он не зависит от природы временного ряда и при правильной настройке параметров способен добиться той же точности, что и второй, так как наличие периодической связи определяет и наличие зависимости данных в окне размера этого периода. Второй метод удобен возможностью добавления в построение

emdedding-слоев экзогенных категориальных признаков, которые в этой задаче также есть (**Номер товара, номер магазина**).

Второй метод по скорости и точности на данной задачи показал значительной лучший результат для экспериментов с нейронными моделями, SMAPE = 12.92%. Первая модель обладает огромной вычислительной затратностью и рассматривалась, по аналогии с экспериментами для VARIMA, на малом объеме данных, где приносила около 16-17% для значения SMAPE. Оба метода представлены в приложении.

3.2 Градиентный бустинг над деревьями

Бустинг является ансамблевым методом, способным скомбинировать несколько слабых моделей в одну сильную, путем последовательного обучение моделей, компенсирующих недостатки друг друга. Данный материал подробно описан в [9]. Градиентный бустинг строит взвешенную сумму базовых алгоритмов:

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x). \tag{31}$$

Построение определяется таким образом - пусть уже построена композиция $a_{N-1}(x)$ из N-1 алгоритма. Тогда следующий базовый алгоритм $b_N(x)$ будет выбираться таким образом, чтобы как можно сильнее уменьшить ошибку:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)) \to \min_{b_N, \gamma_N}, \tag{32}$$

где $L(y, z)$ — некоторая заранее определённая функция потерь. Для этого необходимо определить сдвиги $s_i^{(N)}$, равные направлению антиградиента функции ошибки. Значения этих сдвигов и должен принимать на объектах обучающей выборки новый алгоритм. Необходимо найти базовый алгоритм, приближающий градиент функции потерь на обучающей выборке:

$$b_N(x) = \operatorname*{argmin}_{b \in A} \sum_{i=1}^{\ell} \left(b(x_i) - s_i\right)^2. \tag{33}$$

После того как алгоритм найден, можно подобрать коэффициент γ_n по аналогии с наискорейшим градиентным спуском:

$$\gamma_N = \operatorname*{argmin}_{\gamma \in \mathbb{R}} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma b_N(x_i)) \tag{34}$$

Таким образом, градиентный бустинг осуществляет поиск лучшей функции, восстанавливающей истинную зависимость, в пространстве возможных функций. В случае

решающих деревьев в качестве базовых алгоритмов, композиция обновляется следующим образом:

$$a_N(x) = a_{N-1}(x) + \gamma_N \sum_{j=1}^{J_N} b_{Nj}[x \in R_j], \tag{35}$$

где $j = 1, \ldots, J_n$ — индексы листьев, R_j — соответствующие области разбиения, b_{nj} — значения в листьях. По виду (35) можно судить, добавление в модель одного дерева с J_n листьями равносильно добавлению J_n базовых моделей, являющихся предикатами вида $[x \in R_j]$. Если вместо общего значения γ_N рассматривать γ_{Nj} при каждом предикате, можно подбирать это значение так, чтобы повысить качество общей модели. При этом надобность в компоненте b_{Nj} для минимизации ошибки отпадает:

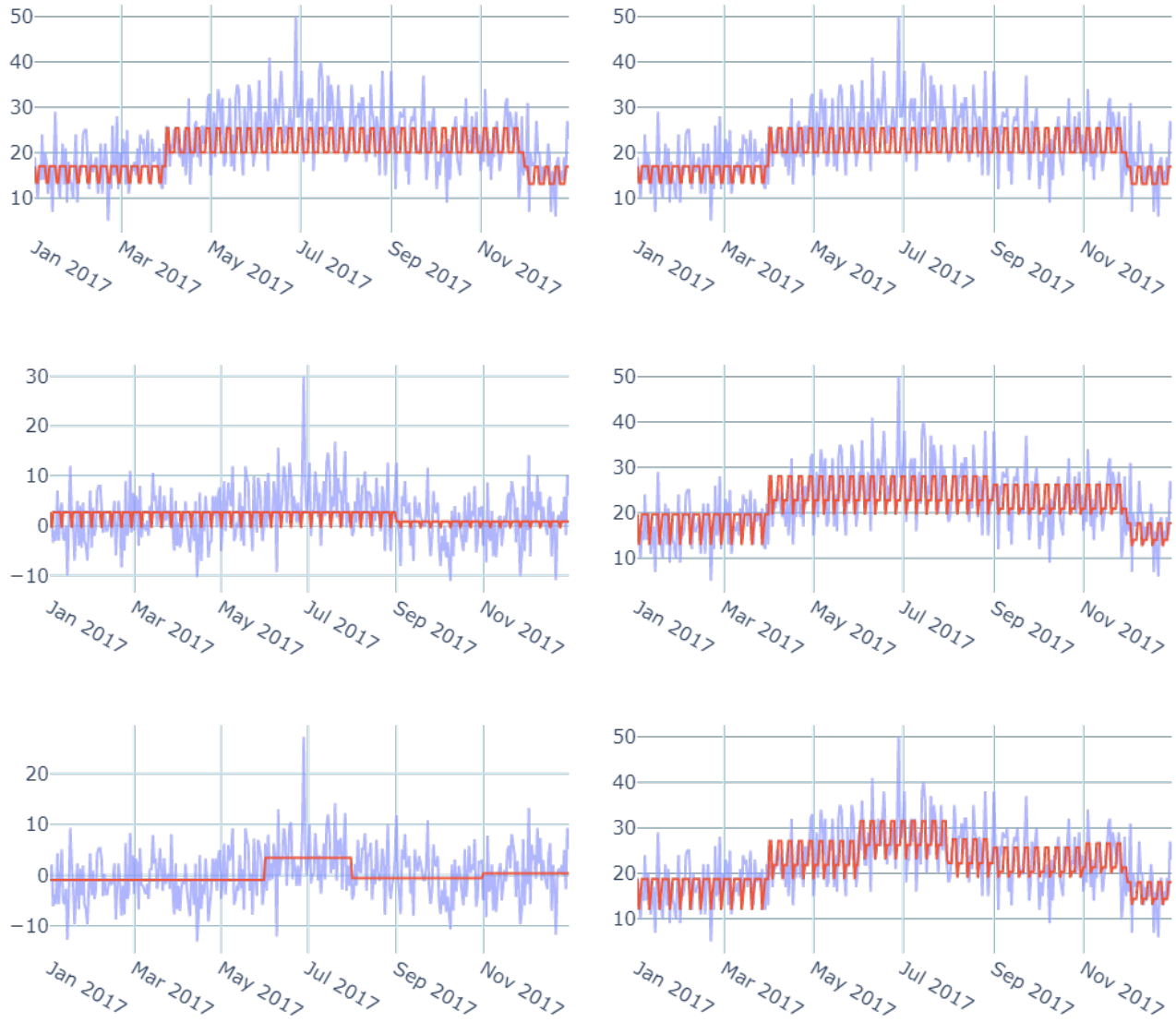
$$\sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x_i) + \sum_{j=1}^{J_N} \gamma_{Nj}[x \in R_j]\right) \to \min_{\{\gamma_{Nj}\}_{j=1}^{J_N}}. \tag{36}$$

Поскольку области R_j не пересекаются данная задача распадается на J_N независимых подзадач:

$$\gamma_{Nj} = \operatorname*{argmin}_{\gamma} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x_i) + \gamma), \qquad j = 1, \ldots, J_N. \tag{37}$$

Определение новой модели заканчивается с определением γ_{Nj}, которые в случае среднеквадратичной ошибки можно вывести аналитически, после чего опять можно продолжать построение модели.

Наибольший результат показала модификация градиентного бустинга LightGBM, реализованная библиотекой одноименной библиотекой, документация по которой представлена в [10]. Настроенная таким образом модель показала результат SMAPE = 12.61%.



(a) Остаточные ошибки и прогнозы первых трех деревьев решений

(b) Прогноз ансамбля после добавления каждого дерева

Рисунок 15 – Построение градиентного бустинга с глубиной дерева равной 3

ЗАКЛЮЧЕНИЕ

Результаты описанных выше методы, реализованных с помощью языка программирования Python, представлены в таблице 3:

Таблица 3 – Оценка времени работы и точности методов

Метод	SMAPE	Время работы, сек
Аналитическая модель	12.61%	17
Линейная регрессия	21.47%	60
Линейная регрессия, полиномиальные признаки, $TS_{i,j}$	13.47%	476
Линейная регрессия, полиномиальные признаки, все данные	40.18%	103
Хольт-Винтерс, недельная сезонность	20.75%	135
ARIMA(7,0,7) с годовой сезонностью	17.68%	4314
ARIMA(2,0,1) с рядами Фурье для 2 сезонностей	13.46%	3298
Нейронная сеть, embedding-слой	12.95%	154
Градиентный бустинг, LGBM	12.61%	49

На основании полученных данных и проведенных различными подходами исследований можно подытожить следующие результаты:

- Предварительный анализ в данной задаче позволил заподозрить синтетическую природу данных, и на основе этого предположения была получена модель, лучше всего объясняющая данные. Также промежуточные результаты анализа на протяжении всего выполнения работы способствовали лучшему пониманию данных и указывали на важные моменты в построении более сложных моделей и оценке их параметров.

- В ходе выполнения работы методами, определёнными для решения задачи многомерного анализа временных рядов, а именно LSTM, MLP, CNN - архитектур нейронных сетей и VARIMA не было достигнуто значительных успехов, по причине огромного количества параллельных процессов и следующих за этим вычислительных затрат. Стоить отметить, что в ходе работы так же было выявлено, что все методы, использующие обучающие данные без учета многомерности процесса, достигли результатов, очень близких к точности аналитического решения.

- Несмотря на высокую корреляцию между всеми $TS_{i,j}$, эту корреляцию можно считать ложной корреляцией, подробное описание которой представлено в работе [11]. Она свидетельствует о том, что рост продаж товаров коррелирует не из-за влияния $TS_{i,j}$

друг на друга, а о наличии общего механизма или зависимости роста продаж для этих товаров.

- Методы машинного обучения обеспечили более высокую точность в сравнении с эконометрическими и статистическими методами. Но это было обеспечено с помощью построения дополнительных признаков, которые были определены, в том числе с помощью упомянутых методов.

СПИСОК ЛИТЕРАТУРЫ

1. Statistical methods for the analysis of observed over time. — URL: https://newonlinecourses.science.psu.edu/stat510 (дата обращения 21.06.2020).

2. SciPy.org, Fourier Transforms. — URL: https://docs.scipy.org/doc/scipy/reference/tutorial/fft.html (дата обращения 21.06.2020).

3. Jack P. NIST/SEMATECH e-Handbook of Statistical Methods, Statistical Methods Group [Электронный ресурс]. — 2012. — URL: http://www.itl.nist.gov/div898/handbook/ (дата обращения 21.06.2020).

4. Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. — URL: http://OTexts.com/fpp2 (дата обращения 21.06.2020).

5. Кареев И. А. Лекции по теории случайных процессов: учебно-методическое пособие. [Электронный ресурс]. — 2016. — https://kms.kpfu.ru/sites/default/files/various/ЭОР/Кареев И.А. Лекции по теории случайных процессов.pdf (дата обращения 21.06.2020)

6. Cleveland R. B., Cleveland W. S., Irma T. STL: A Seasonal-Trend Decomposition Procedure Based on Loess // Journal of Official Statistics; Stockholm. Сер. 6. — 1990. — №1.

7. Lütkepohl H. New Introduction to Multiple Time Series Analysis. — Berlin: Springer-Verlag Berlin Heidelberg, 2005. — 764 с.

8. Seabold, S. Perktold J. statsmodels: Econometric and statistical modeling with python. — URL: https://https://www.statsmodels.org/dev/index.html (дата обращения 21.06.2020).

9. Градиентный бустинг, конспекты лекций по дисциплине «Машинное обучение», Соколов Е. А. [Электронный ресурс]. — 2019. — URL: https://github.com/esokolov/ml-course-hse/blob/master/ (дата обращения 21.06.2020)

10. Документация библиотеки LightGBM. — URL: https://lightgbm.readthedocs.io/en/ (даты обращения 21.06.2020)

11. Симушкин, С. В. Многомерный статистический анализ. — Казань: Издательство КГУ, 2006. — 98 с.

ПРИЛОЖЕНИЕ А

Листинг программного кода

A.1 Предварительный анализ, программный код:

```
import numpy as np
import pandas as pd

import lightgbm as lgb

from sklearn.linear_model import LinearRegression, LassoCV, RidgeCV
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

from keras.layers import Dense, Dropout, Embedding, Input, Reshape,
    Concatenate
from keras.models import Model
import keras.backend as K

from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
from plotly.subplots import make_subplots
import plotly.graph_objs as go
import plotly.offline as py

import matplotlib.pyplot as plt

from statsmodels.tsa import stattools as sts
import statsmodels.api as sm
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.seasonal import STL, seasonal_decompose
from statsmodels.tsa.stattools import acf, pacf
```

```
def SMAPE (forecast, actual):
    """Returns the Symmetric Mean Absolute Percentage Error between two
    Series"""
    masked_arr = ~((forecast==0)&(actual==0))
    diff = abs(forecast[masked_arr] - actual[masked_arr])
    avg = (abs(forecast[masked_arr]) + abs(actual[masked_arr]))/2

    print('SMAPE Error Score: ' + str(round(sum(diff/avg)/len(forecast) *
    100, 2)) + ' %')

PATH = "../input"
```

```
train_df = pd.read_csv(f"{PATH}/train.csv", low_memory=False,
                       parse_dates=['date'])
test_df = pd.read_csv(f"{PATH}/test.csv", low_memory=False,
                      parse_dates=['date'])
sample_sub_df = pd.read_csv(f"{PATH}/sample_submission.csv")

store_df = train_df.copy()
sales_pivoted_df = pd.pivot_table(store_df, index = 'store',
                                  values=['sales','date'], columns = 'item',
    aggfunc = np.mean)

# Scatter plot of avg sales per store
sales_across_store_df = sales_pivoted_df.copy()
sales_across_store_df['avg_sale'] =
sales_across_store_df.apply(lambda r: r.mean(), axis=1)

sales_store_data = go.Scatter(
    x = np.arange(1,11),
    y = sales_across_store_df.avg_sale.values,
    marker = dict(
        size = sales_across_store_df.avg_sale.values,
        color = sales_across_store_df.avg_sale.values,
        colorscale='Viridis',
        showscale=True
    ),
    text = sales_across_store_df.avg_sale.values
)
data = [sales_store_data]

sales_store_layout = go.Layout(
    autosize=True,
    hovermode = 'closest',
    xaxis = dict(
        title = 'Stores',
        ticklen = 10,
        zeroline = False,
        gridwidth = 1,
    ),
    yaxis = dict(
        title = 'Avg Sales',
        ticklen = 10,
        zeroline = False,
        gridwidth = 1,
    ),
    width=1600,
```

```
    height=400,
    margin=dict(l=20, r=20, t=40, b=20),
    font=dict(size=16)
)
fig = go.Figure(data = data, layout = sales_store_layout)
py.iplot(fig, filename='scatter_sales_store')

# Scatter plot of avg sales per item

sales_across_item_df = sales_pivoted_df.copy()
sales_across_item_df.loc[11] = sales_across_item_df.apply(lambda r:r.mean(),
    axis =0)
display(sales_across_item_df)

avg_sales_per_item_across_stores_df = pd.DataFrame(
    data = [[i+1,a] for i,a in enumerate(sales_across_item_df.loc[11:].values
    [0])],
    columns = ['item', 'avg_sale']
)
avg_sales_per_item_across_stores_df.sort_values(by = 'avg_sale', ascending =
    False, inplace = True)

avg_sales_per_item_across_stores_sorted = avg_sales_per_item_across_stores_df
    .avg_sale.values
sales_item_data = go.Bar(
    x = np.arange(0,50),
    y = avg_sales_per_item_across_stores_sorted,
    marker = dict(
        color = avg_sales_per_item_across_stores_sorted,
        colorscale = 'Blackbody',
        showscale = True
    ),
    text = avg_sales_per_item_across_stores_df.item.values
)
data = [sales_item_data]

sales_item_layout = go.Layout(
    autosize= True,
    hovermode= 'closest',
    xaxis= dict(
        title= 'Items',
        ticklen= 55,
        zeroline= False,
        gridwidth= 1,
    ),
```

```
    yaxis=dict(
        title= 'Avg Sales',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
    ),
    width=1600,
    height=400,
    margin=dict(l=20, r=20, t=40, b=20),
    font=dict(size=16),
    showlegend= False
)

fig = go.Figure(data = data, layout = sales_item_layout)
py.iplot(fig,filename = 'scatter_sales_item')

store_item_df = train_df.copy()
store_id = 1
item_id = 10
store_item_df = store_item_df[(store_item_df.store == store_id)&(
    store_item_df.item == item_id)]

plt.plot(store_item_df.sales)

peridiogram = pd.DataFrame(sts.periodogram(store_item_df.sales))
with plt.style.context('seaborn-white'):
    plt.figure(figsize=(11, 3))
    plt.plot(pd.DataFrame(periodogram))
    plt.legend(loc="best")
    plt.axis('tight')
    plt.grid(True)

print('1-st season period : %.8f' %(len(store_item_df.sales)/5))
print('2-nd season period : %.8f' %(len(store_item_df.sales)/261))
print('3-rd season period : %.8f' %(len(store_item_df.sales)/522))

multi_store_item_df = train_df.copy()
multi_store_id = np.arange(1,6)
multi_item_id = np.array([10,20,30,40,50])
multi_store_item_ts_data = []
multi_store_item_df = multi_store_item_df[multi_store_item_df.store.isin(
    multi_store_id)&multi_store_item_df.item.isin(multi_item_id)]

for st, i in zip(multi_store_id, multi_item_id):
    flt = multi_store_item_df[(multi_store_item_df.store == st)&(
```

```
    multi_store_item_df.item == i)]
138     multi_store_item_ts_data.append(go.Scatter(x = flt.date, y = flt.sales.
    rolling(10, win_type='triang').sum(),
139                                                name = "Store:" + str(st) + ",
    Item:" + str(i)))
140
141 multy_store_layout = go.Layout(
142     autosize= True,
143     hovermode= 'closest',
144     xaxis= dict(
145         title= 'Date',
146         ticklen= 55,
147         zeroline= False,
148         gridwidth= 1,
149         gridcolor='LightBlue'
150     ),
151     yaxis=dict(
152         title= 'Sales',
153         ticklen= 10,
154         zeroline= False,
155         gridwidth= 1,
156         gridcolor='LightBlue'
157     ),
158     width=1000,
159     height=500,
160     margin=dict(l=20, r=20, t=40, b=20),
161     paper_bgcolor='rgba(0,0,0,0)',
162     plot_bgcolor='rgba(0,0,0,0)',
163     font=dict(size=16),
164 )
165
166 fig = go.Figure(data = multi_store_item_ts_data, layout = multy_store_layout)
167 py.iplot(fig)
168
169 # Extract month, day of week, year info from 'date' field
170
171 def expand_df(df):
172     data = df.copy()
173     data['day'] = data.index.day
174     data['month'] = data.index.month
175     data['year'] = data.index.year
176     data['dayofweek'] = data.index.dayofweek
177     return data
178
179 data = expand_df(train_df)
```

```
display(data)

grand_avg = data.sales.mean()

print(f"The grand average of sales in this dataset is {grand_avg:.4f}")

years = np.arange(2013, 2019)
day_of_week =  np.arange(1, 8)
month = np.arange(1, 13)

# Annual change

agg_year_item = pd.pivot_table(data, index='year', columns='item',
                               values='sales', aggfunc=np.mean).values
agg_year_store = pd.pivot_table(data, index='year', columns='store',
                                values='sales', aggfunc=np.mean).values

plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.plot(years[:-1], agg_year_item)
plt.title("Items")
plt.xlabel("Year")
plt.ylabel("Sales")
plt.subplot(122)
plt.plot(years[:-1],agg_year_store)
plt.title("Stores")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.show()

# Relative annual data

plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.plot(agg_year_item / agg_year_item.mean(0)[np.newaxis])
plt.title("Items")
plt.xlabel("Year")
plt.ylabel("Relative Sales")
plt.subplot(122)
plt.plot(agg_year_store / agg_year_store.mean(0)[np.newaxis])
plt.title("Stores")
plt.xlabel("Year")
plt.ylabel("Relative Sales")
plt.show()

```

```
# Monthly relative data

agg_month_item = pd.pivot_table(data, index='month', columns='item',
                                values='sales', aggfunc=np.mean).values
agg_month_store = pd.pivot_table(data, index='month', columns='store',
                                 values='sales', aggfunc=np.mean).values

plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.plot(agg_month_item / agg_month_item.mean(0)[np.newaxis])
plt.title("Items")
plt.xlabel("Month")
plt.ylabel("Relative Sales")
plt.subplot(122)
plt.plot(agg_month_store / agg_month_store.mean(0)[np.newaxis])
plt.title("Stores")
plt.xlabel("Month")
plt.ylabel("Relative Sales")
plt.show()

# Day of week relative data

agg_dow_item = pd.pivot_table(data, index='dayofweek', columns='item',
                              values='sales', aggfunc=np.mean).values
agg_dow_store = pd.pivot_table(data, index='dayofweek', columns='store',
                               values='sales', aggfunc=np.mean).values

plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.plot(agg_dow_item / agg_dow_item.mean(0)[np.newaxis])
plt.title("Items")
plt.xlabel("Day of Week")
plt.ylabel("Relative Sales")
plt.subplot(122)
plt.plot(agg_dow_store / agg_dow_store.mean(0)[np.newaxis])
plt.title("Stores")
plt.xlabel("Day of Week")
plt.ylabel("Relative Sales")
plt.show()

store_item_table = pd.pivot_table(data, index='store', columns='item',
                                  values='sales', aggfunc=np.mean)
display(store_item_table)

# Monthly pattern
```

```
month_table = pd.pivot_table(data, index='month', values='sales', aggfunc=np.mean)
month_table.sales /= grand_avg

# Day of week pattern
dow_table = pd.pivot_table(data, index='dayofweek', values='sales', aggfunc=np.mean)
dow_table.sales /= grand_avg

# Yearly growth pattern
year_table = pd.pivot_table(data, index='year', values='sales', aggfunc=np.mean)
year_table /= grand_avg

annual_sales_avg = year_table.values.squeeze()

p1 = np.poly1d(np.polyfit(years[:-1], annual_sales_avg, 1))
p2 = np.poly1d(np.polyfit(years[:-1], annual_sales_avg, 2))

plt.figure(figsize=(8,6))
plt.plot(years[:-1], annual_sales_avg, 'ko')
plt.plot(years, p1(years), 'C0-')
plt.plot(years, p2(years), 'C1-')
plt.xlim(2012.5, 2018.5)
plt.title("Relative Sales by Year")
plt.ylabel("Relative Sales")
plt.xlabel("Year")
plt.show()

print(f"2018 Relative Sales by Degree-1 (Linear) Fit = {p1(2018):.4f}")
print(f"2018 Relative Sales by Degree-2 (Quadratic) Fit = {p2(2018):.4f}")

# We pick the quadratic fit
annual_growth = p2

def Analytical_solution(test, submission):
    submission[['sales']] = submission[['sales']].astype(np.float64)
    for _, row in test.iterrows():
        dow, month, year = row.name.dayofweek, row.name.month, row.name.year
        item, store = row['item'], row['store']
        base_sales = store_item_table.at[store, item]
        mul = month_table.at[month, 'sales'] * dow_table.at[dow, 'sales']
        pred_sales = base_sales * mul * annual_growth(year)
        submission.at[row['id'], 'sales'] = pred_sales
    return submission
```

```
312
313 Analytical_solution_pred = Analytical_solution(test_df, sample_sub_df.copy())
```

A.2 Статистические методы, программный код:

```
1
2 train_df['Year'] = pd.DatetimeIndex(train_df.Date).year - 2013
3 train_df['Month'] = pd.DatetimeIndex(train_df.Date).month
4 train_df['Weekday'] = pd.DatetimeIndex(train_df.Date).weekday
5 train_df['Day_in_total'] = pd.DatetimeIndex(train_df.Date).dayofyear + (pd.
    DatetimeIndex(train_df.Date).year-2013)*365
6 train_df.loc[train_df['Year']>3, 'Day_in_total'] +=1
7
8 test_df['Year'] = pd.DatetimeIndex(test_df.Date).year - 2013
9 test_df['Month'] = pd.DatetimeIndex(test_df.Date).month
10 test_df['Weekday'] = pd.DatetimeIndex(test_df.Date).weekday
11 test_df['Day_in_total'] = pd.DatetimeIndex(test_df.Date).dayofyear + (pd.
    DatetimeIndex(test_df.Date).year-2013)*365
12 test_df.loc[train_df['Year']>3, 'Day_in_total'] +=1
13
14 train_df.drop(columns = 'Date', inplace = True)
15 test_df.drop(columns = 'Date', inplace = True)
16
17 # Trend with linear regression
18
19 RidgeLR = RidgeCV(normalize=True)
20 X_train = np.arange(1,1827)
21 y_train = train_df[(train_df['Store']==1)&(train_df['Item']==1)].Sales.values
22 RidgeLR.fit(X_train.reshape(-1, 1), y_train)
23
24 Prediction = RidgeLR.predict(X_train.reshape(-1, 1))
25 Prediction
26
27 # Polynomial Features
28
29 poly = PolynomialFeatures(degree=3)
30 poly.fit(train_df.drop(columns='Sales'))
31 train_poly = poly.transform(train_df.drop(columns='Sales'))
32 test_poly = poly.transform(test_df.drop(columns='ID'))
33
34 train_poly_df = pd.DataFrame(train_poly, columns = poly.get_feature_names())
35 test_poly_df = pd.DataFrame(test_poly, columns = poly.get_feature_names())
36
37 train_poly_df['Sales'] = train_df.Sales
38
```

```
# Ridge-regression with CV

X_train, y_train = train_poly_df[(train_poly_df['x0']==1)&(train_poly_df['x1']==1)&(train_poly_df['x5']<1462)].drop(columns = ['Sales']), \
                   train_poly_df[(train_poly_df['x0']==1)&(train_poly_df['x1']==1)&(train_poly_df['x5']<1462)].Sales
X_valid, y_valid = train_poly_df[(train_poly_df['x0']==1)&(train_poly_df['x1']==1)&(train_poly_df['x5']>1461)&(train_poly_df['x3']<4)].drop(columns = ['Sales']), \
                   train_poly_df[(train_poly_df['x0']==1)&(train_poly_df['x1']==1)&(train_poly_df['x5']>1461)&(train_poly_df['x3']<4)].Sales

RidgeLR = RidgeCV(normalize=True)
RidgeLR.fit(X_train, y_train)
Prediction = RidgeLR.predict(X_valid)

# Ridge-regression for test data

for s in range(1, 11):
    for i in range(1,51):
        X_train = train_poly_df[(train_poly_df['x0']==s)&(train_poly_df['x1']==i)].drop(columns = ['Sales'])
        y_train = train_poly_df[(train_poly_df['x0']==s)&(train_poly_df['x1']==i)].Sales
        X_test =  test_poly_df[(test_poly_df['x0']==s)&(test_poly_df['x1']==i)]
        RidgeLR = RidgeCV(normalize=True)
        RidgeLR.fit(X_train, y_train)
        Prediction = RidgeLR.predict(X_test)
        test_df.loc[(test_df['Store']==s)&(test_df['Item']==i),'Sales'] = Prediction

# All in one

LassoLR = LassoCV(normalize=True)
LassoLR.fit(train_poly_df, train_df.Sales)
prediction = LassoLR.predict(test_poly_df)
```

```
## Exponential smoothing

def exponential_smoothing(series, alpha):
    result = [series[0]] # first value is same as series
    for n in range(1, len(series)):
        result.append(alpha * series[n] + (1 - alpha) * result[n-1])
    return result

```

```
data = train_df[(train_df['Store']==1)&(train_df['Item']==1)].drop(columns =
    ['Sales'])
data_y = train_df[(train_df['Store']==1)&(train_df['Item']==1)].Sales

# matplotlib
with plt.style.context('seaborn-white'):
            plt.figure(figsize=(15, 3))
            plt.plot(exponential_smoothing(data_y, 0.2), label = 'Smoothed')
            plt.plot(data_y, label = 'Real', alpha = 0.5)
            plt.legend(loc="best", fontsize = 'x-large')
            plt.axis('tight')
            plt.grid(True)

## Holt-Winters model

for i in range(1,51):
    for s in range(1, 11):
        X_train = train_df[(train_df['Store']==s)&(train_df['Item']==i)].
    Sales
        model = ExponentialSmoothing(X_train,
                                     trend='add',
                                     seasonal_periods=7,
                                     seasonal='add').fit()

        test_df.loc[(test_df['Store']==s)&(test_df['Item']==i),'Sales'] = np.
    array(model.forecast(90))
```

```
# ARIMA(2,0,1) with Fourier Term

train_df['sin365'] = np.sin(2 * np.pi *pd.DatetimeIndex(train_df.Date).
    dayofyear / 365)
train_df['cos365'] = np.cos(2 * np.pi * pd.DatetimeIndex(train_df.Date).
    dayofyear / 365)
train_df['sin365_2'] = np.sin(4 * np.pi * pd.DatetimeIndex(train_df.Date).
    dayofyear / 365)
train_df['cos365_2'] = np.cos(4 * np.pi * pd.DatetimeIndex(train_df.Date).
    dayofyear / 365)

test_df['sin365'] = np.sin(2 * np.pi *pd.DatetimeIndex(test_df.Date).
    dayofyear / 365)
test_df['cos365'] = np.cos(2 * np.pi * pd.DatetimeIndex(test_df.Date).
    dayofyear / 365)
test_df['sin365_2'] = np.sin(4 * np.pi * pd.DatetimeIndex(test_df.Date).
    dayofyear / 365)
test_df['cos365_2'] = np.cos(4 * np.pi * pd.DatetimeIndex(test_df.Date).
    dayofyear / 365)
```

```
12
13 for i in range(1,51):
14     for s in range(1,11):
15         sample = np.array(train_df[(train_df['Item']==i)&(train_df['Store']==
   s)].Sales)
16         sample_exog = np.array(train_df.loc[(train_df['Item']==i)&(train_df['
   Store']==s), ['sin365', 'cos365', 'sin365_2','cos365_2']])
17         target_exog = np.array(test_df.loc[(test_df['Item']==i)&(test_df['
   Store']==s), ['sin365', 'cos365', 'sin365_2','cos365_2']])
18
19         mod = sm.tsa.arima.ARIMA(sample, order=(2, 0, 1), seasonal_order
   =(1,0,1,7), exog = sample_exog)
20         res = mod.fit()
21         test_df.loc[(test_df['Item']==i)&(test_df['Store']==s), 'sales'] =
   res.forecast(90, exog = target_exog)
22
23 # ARIMA(7,0,7)
24 res_SMAPE = 0
25 for i in range(1,51):
26     for s in range(1,11):
27         sample = np.array(train_df[(train_df['Item']==i)&(train_df['Store']==
   s)].Sales)
28
29         mod = sm.tsa.arima.ARIMA(sample, order=(7, 0, 7))
30         res = mod.fit()
31         test_df.loc[(test_df['Item']==i)&(test_df['Store']==s), 'sales'] =
   res.forecast(90)
```

A.3 Методы машинного обучения, программный код:

```
1 # Embedding neural network
2
3 train_data = pd.read_csv("../input/train.csv", parse_dates=True, index_col
   =0)
4 y_train = train_data.iloc[:,-1].values
5 train_data.drop('sales', 1, inplace=True)
6 train_data['Year'] = train_data.index.year-train_data.index.year.min()
7 train_data['Month'] = train_data.index.month
8 train_data['Day'] = train_data.index.day
9 train_data['DOW'] = train_data.index.dayofweek
10
11 x_train, x_val, y_train, y_val = train_test_split(train_data, y_train,
   test_size=.1, random_state=0, shuffle = True)
12
13 cat_vars = list(train_data.columns)
```

```
cat_vars.remove('Year')
cont_vars = ['Year']

X_train = []
X_val = []
X_train.append(x_train[cont_vars].astype('float32').values)
X_val.append(x_val[cont_vars].astype('float32').values)
for cat in cat_vars:
    X_train.append(x_train[cat].values)
    X_val.append(x_val[cat].values)

cat_sizes = {}
cat_embsizes = {}
for cat in cat_vars:
    cat_sizes[cat] = train_data[cat].nunique()
    cat_embsizes[cat] = min(50, cat_sizes[cat]//2+1)

test_data = pd.read_csv("../input/test.csv", parse_dates=True, index_col =1)
test_data['Year'] = test_data.index.year-train_data.index.year.min()
test_data['Month'] = test_data.index.month
test_data['Day'] = test_data.index.day
test_data['DOW'] = test_data.index.dayofweek


def custom_smape(x, x_):
    return K.mean(2*K.abs(x-x_)/(K.abs(x)+K.abs(x_)))

y = Input((len(cont_vars),), name='cont_vars')
ins = [y]
concat = [y]
for cat in cat_vars:
    x = Input((1,), name=cat)
    ins.append(x)
    x = Embedding(cat_sizes[cat]+1, cat_embsizes[cat], input_length=1)(x)
    x = Reshape((cat_embsizes[cat],))(x)
    concat.append(x)
y = Concatenate()(concat)
y = Dense(100, activation= 'relu')(y)
# y = Dense(100, activation= 'relu')(y)
y = Dense(1)(y)
model = Model(ins, y)
model.summary()
model.compile('adadelta', custom_smape)

model.fit(X_train, y_train, 64, 2, validation_data=[X_val, y_val])
```

```
59
60 test_preds = model.predict(X_test)
61
62 # Gradient descent
63
64 def add_agg(merged_df, gr_cols, new_col_name, incr_yr):
65     agg1 = train_df.groupby(gr_cols)['target'].agg('mean').reset_index()
66     cols2 = gr_cols.copy()
67     cols2.append(new_col_name)
68     agg1.columns = cols2
69     if incr_yr:
70         agg1['year']+=1
71     merged_df = pd.merge(merged_df, agg1, how='left', left_on=gr_cols,
       right_on=gr_cols)
72     return merged_df
73
74 input_dir  = os.path.join(os.pardir, 'input')
75 train_df   = pd.read_csv(os.path.join(input_dir, 'train.csv'), nrows=None)
76 test_df    = pd.read_csv(os.path.join(input_dir, 'test.csv'), nrows=None)
77
78 merged_df = pd.concat([train_df, test_df], sort=False)
79
80 merged_df['date']  = pd.to_datetime(merged_df['date'],infer_datetime_format=
       True)
81 merged_df['year']  = merged_df['date'].dt.year
82 merged_df['month']  = merged_df['date'].dt.month
83 merged_df['day']   = merged_df['date'].dt.dayofweek
84 merged_df.drop('date', axis=1, inplace=True)
85
86 train_df=pd.DataFrame(merged_df[merged_df.sales.notna()].values)
87 train_df.columns=merged_df.columns
88 train_df['target']=train_df['sales']
89
90 merged_df = add_agg(merged_df,['item','store','year'], 'tsy', 1)
91 merged_df['sales']/=merged_df['tsy']
92 merged_df = merged_df[merged_df.year>2013]
93 tsy=merged_df.pop('tsy')
94
95 ID=merged_df[merged_df.id.notna()]['id']
96 target=merged_df[merged_df.sales.notna()]['sales']
97 merged_df.drop(['id','sales'], axis=1, inplace=True)
98 len_train=target.shape[0]
99
100 params = {
101     'nthread': 10,
```

```
    'max_depth': 8,
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'regression_l1',
    'metric': 'mape', # this is abs(a-e)/max(1,a)
    'num_leaves': 31,
    'learning_rate': 0.25,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'lambda_l1': 0.06,
    'lambda_l2': 0.1,
    'verbose': -1
}

num_folds = 5
test_x = merged_df[len_train:].values
all_x = merged_df[:len_train].values
all_y = target.values
oof_preds = np.zeros([all_y.shape[0]])
sub_preds = np.zeros([test_x.shape[0]])
feature_importance_df = pd.DataFrame()
folds = KFold(n_splits=num_folds, shuffle=True, random_state=345665)
for n_fold, (train_idx, valid_idx) in enumerate(folds.split(all_x)):
    train_x, train_y = all_x[train_idx], all_y[train_idx]
    valid_x, valid_y = all_x[valid_idx], all_y[valid_idx]
    lgb_train = lgb.Dataset(train_x,train_y)
    lgb_valid = lgb.Dataset(valid_x,valid_y)

    gbm = lgb.train(params, lgb_train, 1000,
        valid_sets=[lgb_train, lgb_valid],
        early_stopping_rounds=100, verbose_eval=100)
    oof_preds[valid_idx] = gbm.predict(valid_x, num_iteration=gbm.
    best_iteration)
    sub_preds[:] += gbm.predict(test_x, num_iteration=gbm.best_iteration) /
    folds.n_splits
    valid_idx += 1
    importance_df = pd.DataFrame()
    importance_df['feature'] = merged_df.columns
    importance_df['importance'] = gbm.feature_importance()
    importance_df['fold'] = n_fold + 1
    feature_importance_df = pd.concat([feature_importance_df, importance_df],
    axis=0)
e = 2 * abs(all_y - oof_preds) / ( abs(all_y)+abs(oof_preds) )
e = e.mean()
```

```
144 print('Full validation score %.4f' %e)
145
146 pred = (sub_preds * tsy[len_train:] ).astype(np.float32)
147 out_df = pd.DataFrame({'id': ID.astype(np.int32), 'sales': pred})
```