Санкт-Петербургский государственный университет

Шульга Валентин Александрович

Магистерская диссертация

Сравнительный анализ алгоритмов машинного обучения в задачах исследования фондового рынка

Направление 01.04.02 «Прикладная математика и информатика»
Основная образовательная программа ВМ.5759.2018 «Цифровая экономика»

Научные руководители:

Старший преподаватель кафедры Технологии программирования

Стученков Александр Борисович

профессор кафедры МЭС

Смирнов Николай Васильевич

Санкт-Петербург

2020

Содержание

Используемые сокращения и определения

Приведём основные понятия и термины, используемые в данной работе:

RNN – рекуррентная нейронная сеть, это вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки.

LSTM – Долгая краткосрочная память. Разновидность архитектуры рекуррентных нейронных сетей.

Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше.

Синапс - связь между несколькими нейронами.

Цена закрытия (closing price) –цена последней сделки, зарегистрированная при **закрытии** срочной биржи по окончании рабочего дня**.**

Объём (volume) **-** технический индикатор, отражающий реальный объём (оборот) торгов по количеству купленных, проданных к примеру, акций за выбранный промежуток времени.

Эпоха - один проход по всему набору данных, используемый для разделения обучения на отдельные фазы, важно для ведения логов и периодической оценки.

МО- Машинное обучение

НС – Нейронная сеть

LR – Linear Regression

Введение

Прогнозирование фондового рынка - это попытка определить будущую стоимость акций компании или другого финансового инструмента, торгуемого на бирже. Успешное прогнозирование будущей цены акций может принести хорошую прибыль.

Гипотеза об эффективном рынке говорит, что цены на акции отражают всю имеющуюся в настоящее время информацию, и любые изменения цен, которые не основаны на недавно выявленной информации, по своей сути непредсказуемы. Другие не согласны, и те, у кого есть эта точка зрения, обладают бесчисленными методами и технологиями, которые предположительно позволяют им получать информацию о будущих ценах.

Машинное и в частности глубокое обучение становятся новой эффективной стратегией, которую для увеличения доходов используют многие инвестиционные фонды. Изменения в сфере финансов происходят нелинейно, и иногда может показаться, что цены на акции формируются совершенно случайным образом.

Традиционные методы временных рядов, такие как модели ARIMA и GARCH эффективны, когда ряд является стационарным - его основные свойства со временем не изменяются [6]. А для этого требуется, чтобы ряд был предварительно обработан приведён к стационарности. Однако главная проблема возникает при реализации этих моделей в реальной торговой системе, так как при добавлении новых данных стационарность не гарантируется.

В современном мире всё с большей остротой проявляется интерес к качественному прогнозированию финансовых рынков. Это связано с быстрым развитием высоких технологий и, соответственно, с появлением новых инструментов анализа данных. Однако тот технический анализ, которым привыкли пользоваться большинство участников рынка, неэффективен. А

именно, прогнозы на основе экспоненциальных скользящих средних, осцилляторов и прочих индикаторов не выдают хороший результат, так как экономика бывает иррациональна, движима иррациональностью людей.

В последние годы, у финансовых аналитиков вызывает большой интерес так называемые искусственные нейронные сети – это математические модели, а также их программные или аппаратные реализации, построенные по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма. Такое определение возникло при изучении процессов, протекающих в мозге при мышлении, и при попытке смоделировать эти процессы. Впоследствии эти модели стали использовать в практических целях, как правило, в задачах прогнозирования.

Нейронные сети не программируются в привычном смысле этого слова, они обучаются. Возможность обучения – одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение [5].

Способности нейронной сети к прогнозированию напрямую следуют из её способности к обобщению и выделению скрытых зависимостей между входными и выходными данными. После обучения сеть способна предсказывать будущие значения некой последовательности на основе нескольких предыдущих значений и/или каких-то существующих в настоящий момент факторов.

Следует отметить, что прогнозирование возможно только тогда, когда предыдущие изменения действительно в какой-то степени предопределяют будущие. Например, прогнозирование котировок акций на основе котировок за прошлую неделю может оказаться успешным, тогда как прогнозирование результатов завтрашней лотереи на основе данных за последние 50 лет почти наверняка не даст никаких результатов.

Существует две методики прогнозирования цен акций:

- **Фундаментальный анализ** — в этом случае аналитики оценивают информацию, которая больше относится к компании, чьи акции торгуются на бирже, нежели к самим акциям. Решения о тех или иных действиях на рынке принимаются на основе анализа предыдущей деятельности компании, прогнозах выручки и прибыли и так далее.
- **Технический анализ** — в данном случае рассматривается поведение цены акций и выявляются его разнообразные паттерны (используется анализ временных рядов).

В случае применения методов машинного обучения для обработки торговых данных, чаще используют именно метод технического анализа - цель заключается в том, чтобы понять, сможет ли алгоритм точно определять паттерны поведения акции во времени. Но, машинное обучение может использоваться и для оценки и прогнозирования результатов деятельности компании для дальнейшего использования при фундаментальном анализе.

Постановка задачи

Прогнозирование фондового рынка связано не просто с индивидуальным извлечением прибыли конкретными игроками, а ещё и с общим увеличением эффективности и стабильности финансовой отрасли и экономики в целом путем увеличения ликвидности.

Целью данной работы является исследование возможностей применения алгоритмов машинного обучения, в частности, нейросетевых технологий, в вопросах прогнозирования поведения финансовых инструментов, трендов и движения цен, а также последующее проведение сравнительного анализа эффективности применения рассмотренных алгоритмов.

Для достижения поставленной цели требуется решить следующие **задачи**:

• Проведение анализа различных методов машинного обучения, архитектур нейросетей, библиотек и инструментов, с целью их дальнейшего применения в работе.

• Поиск, предобработка, оптимизация и нормализация данных для обучения и тестирования.

• Программная реализация модели.

• Визуальное представление и сравнение результатов работы алгоритмов.

В результате будет разработана исследовательская программа на языке Python, решающая поставленную задачу.

Обзор литературы

При написании данной работы были использованы научные статьи, интернет-ресурсы и другие материалы по вопросам прогнозирования фондового рынка, проведения анализа существующих методов прогнозирования и влияния технического анализа на прогнозирование данных:

Прогнозирование индекса фондового рынка с использованием искусственной нейронной сети описывается в книге [6]. В статье [14] рассматриваются основы LSTM-сети, выводится математическая формулировка нейросети. Схематически изображается как работает рекуррентная нейронная сеть.

Основные определения и характеристики фреймворков Tensorflow и Keras описываются в ресурсах [2], [7] соответственно. В книге [11] описываются основы и устройство Tensorflow, проводится обучение по нему, и рассматриваются примеры построения моделей. В статье [4] автор рассказывает о всех нюансах работы с временными рядами в Python, вводится понятие стационарности, объясняется обоснованность применения линейной регрессии для анализа данных.

Детальное изложение основных определений нейронной сети и описания всех внутренних процессов представлено в [2]. Изучение ключевых концепций нейросетей, включая и современные техники глубокого обучения, а также приобретение фундамента для использования нейронных сетей и глубокого обучения в подходе к решению собственных задач помогает книга [15].

О том, каких результатов можно достичь с помощью рекуррентной нейросети (RNN), чем она занимается, в чём её главные преимущества и недостатки рассказывает статья [8]. Детальное описание того, как создавалась с нуля сеть LSTM, что этому предшествовала, основные преимущества такого подхода в задачах прогнозирования описывается в книге [11].

В книге [3] описывается возможность улучшения классификации путём комбинирования классификации случайно сгенерированных тренировочных наборов, и в частности, описывается алгоритм «Случайный лес».

ГЛАВА 1. Анализ нейросетевых технологий, применяемых для решения задачи

В данной главе рассматриваются две архитектуры рекуррентной нейронной сети - LSTM и MLP. Выбор сделан в пользу именно этих архитектур, поскольку необходимо регулярно обращаться к временному ряду (история движения цен) и учитывать долгосрочный контекст. По итогу главы будет осуществлён подробный разбор каждой выбранной архитектуры сети, описаны плюсы и минусы данных нейросетей.

1.1. Рекуррентная нейронная сеть (RNN)

Последовательная модель обычно предназначена для преобразования входной последовательности (например, временного ряда) одной предметной области в выходную последовательность (в той же или другой предметной области). **Рекуррентная нейронная сеть (Recurrent Neural Network, RNN)** [6], подходит для этой цели. RNN создавалась с возможностью обработки длинных последовательных данных и решения задач с распределением контекста во времени.

Модель обрабатывает один элемент в последовательности на одном временном шаге. После вычисления обновлённое состояние передается на следующий шаг во времени, чтобы облегчить вычисление следующего элемента.

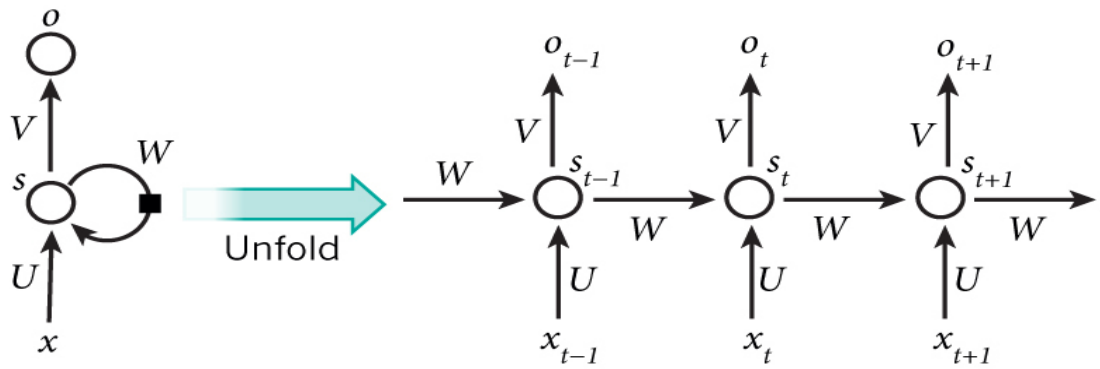

Рис.1. Рекуррентная нейронная сеть с 1 скрытым элементом (слева) и её раскрывающейся версией во времени (справа). Развёрнутая версия показывает, что происходит во времени: S_{t-1}, S_t и S_{t+1} - это одна и та же единица с различными состояниями на разных временных шагах $t - 1$, t и $t + 1$.

Но есть вероятность того, что простые персептронные сети, линейно объединяющие текущий элемент ввода и последний элемент вывода, могут с лёгкостью потерять долгосрочные зависимости.

Для решения проблем забывания, был создан специальный нейрон с более сложной внутренней структурой для запоминания долгосрочного контекста, называемого ячейкой **Longshort Term Memory (LSTM)** [14]. Он достаточно умён, чтобы узнать, как долго должен запоминать старую информацию, когда использовать новые данные и как объединить старую память с новым входом. В LSTM цепь содержит 4 слоя и взаимодействие происходит между ними особенным способом.

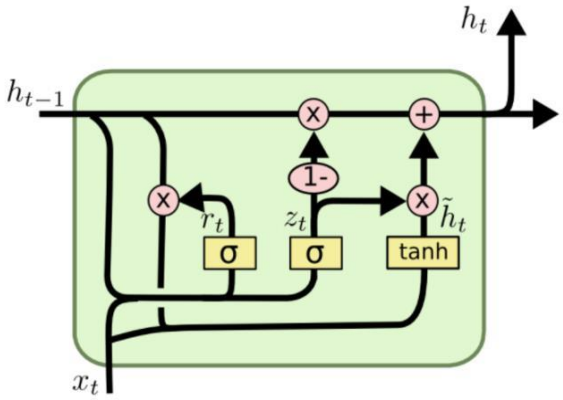

Рис.2.Структура LSTM нейрона

Такая архитектура рекуррентной нейронной сети не использует функцию активации, тем самым значение об ошибках не размываются со временем, и находятся в памяти сети. Данная архитектура вычисляет, насколько поданное значение используется в выходном значении, забывая старое значение, если появится новое лучшее значение, тем самым реализуя более интеллектуальное использование своей памяти.

Опишем главную идею LSTM. Самый основной компонент LSTM – это состояние ячейки (cell state) – горизонтальная линия, проходящая по верхней части схемы, проходящая через всю цепочку и принимающая участие лишь в нескольких линейных преобразованиях. Информация об ошибке может легко течь по ней, не подвергаясь изменениям.

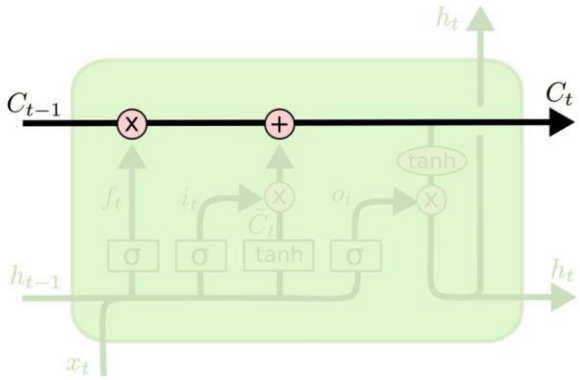

Рис.3.Горизонтальная линия LSTM

Но тем временем, LSTM может удалять информацию из состояния ячейки; этот процесс регулируется структурами, называемыми фильтрами (gates). Фильтры могут позволять пропускать информацию на основании некоторых условий. Они состоят из слоя сигмоидальной нейронной сети и операции поточечного умножения.

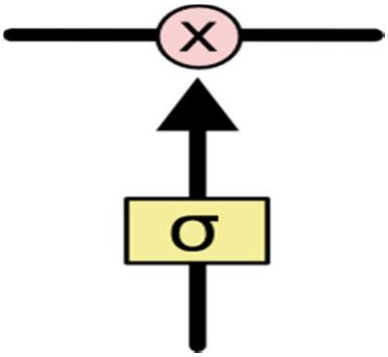

Рис.4.Фильтры в LSTM

Сигмоидальный слой возвращает числа от 0 до 1, обозначающие, какую долю каждого блока информации следует пропустить далее по сети, здесь означает “не пропускать ничего”, 1- “пропустить всё”. В LSTM 3 таких фильтра, позволяющих защищать и контролировать состояние ячейки.

1.2. Многослойный персептрон (MLP)

Такая архитектура нейронной сети используется сейчас наиболее часто. Она была предложена Румельхартом в 1986 году и является частным случаем перцептрона Розенблатта, в котором один алгоритм обратного распространения ошибки обучает все слои. Предлагается почти во всех учебниках по нейронным сетям.

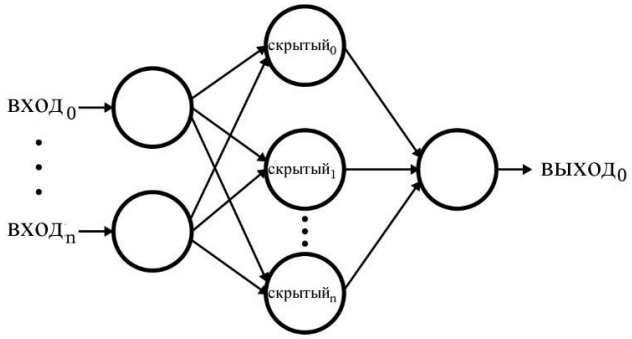

Рис.5.Архитектура рассматриваемой нейронной сети

Каждый элемент сети строит взвешенную сумму своих входов с поправкой в виде слагаемого и затем пропускает эту величину активации через передаточную функцию, и тем самым получается выходное значение этого элемента. Элементы организованы в послойную топологию с прямой передачей сигнала. Такую сеть легко можно интерпретировать как модель вход-выход, в которой веса и пороговые значения (смещения) являются свободными параметрами модели [14]. Такая сеть может моделировать функцию практически любой степени сложности, причем число слоёв и число элементов в каждом слое определяют сложность функции. Определение числа промежуточных слоёв и числа элементов в них является важным вопросом при конструировании MLP.

Количество входных и выходных элементов определяется условиями задачи. Сложность может заключаться в том, какими входными значениями воспользоваться, а какими не стоит. Сейчас отметим, что все входные

переменные выбраны интуитивно и являются значимыми. Дилемма в том, сколько использовать промежуточных слоёв и элементов в них, совершенно неизвестна. В качестве начального приближения можно взять один промежуточный слой, а число элементов в нём сделать равным полусумме числа входных и выходных элементов.

Обучение многослойного персептрона

После того, как выявлено число слоёв и элементов в каждом из них, есть смысл искать значения для весов и порогов сети, минимизирующую ошибку прогноза, полученного сетью, использовать *алгоритмы обучения*. С применением собранных исторических данных веса и пороговые значения автоматически корректируются с целью минимизации этой ошибки.

В итоге такой процесс представим в виде подгонки модели, реализующееся сетью, к имеющимся обучающим данным. Ошибка для конкретной конфигурации сети формируется путём прогона через сеть всех имеющихся наблюдений и сравнения реально выдаваемых выходных значений с желаемыми (целевыми) значениями. Все такие разности суммируются в *функцию ошибок*, значение которой и есть ошибка сети.

В качестве функции ошибок часто применяют сумма квадратов ошибок, т.е. когда все ошибки выходных элементов для всех наблюдений возводятся в квадрат и после суммируются. В традиционном моделировании (линейном) можно алгоритмически указать конфигурацию модели, выдающую абсолютный минимум для указанной ошибки.

Цена, которую необходимо заплатить за более расширенные (нелинейные) возможности моделирования с помощью нейросетей, заключается в том, что, корректируя сеть с целью минимизировать ошибку, никогда не сможем быть уверены в том, что невозможно достичь ещё меньшей ошибки.

В таких рассмотрениях очень полезным будет являться определение поверхности ошибок. Каждому из весов и порогов сети (т.е. свободных параметров модели; их общее число обозначим через *N*) будет соответствовать одно измерение в многомерном пространстве. N+1-е измерение равно ошибке сети. Для любых возможных сочетаний весов соответствующую ошибку сети можно изобразить точкой в N+1-мерном пространстве, и все такие точки образуют там некоторую поверхность - *поверхность ошибок*. Целью обучения нейронной сети будет являться задача нахождение на этой многомерной поверхности самой низкой точки.

В случае линейной модели с суммой квадратов в качестве функции ошибок такая поверхность ошибок будет представима в виде параболоида (квадрика) - гладкой поверхности, похожей на часть поверхности сферы, с единственным минимумом. Здесь локализовать такой минимум очень просто.

В случае нейронной сети поверхность ошибок имеет более сложное строение и обладает рядом негативных свойств, например, иметь локальные минимумы (точки, самые низкие в некоторой своей окрестности, но лежащие выше глобального минимума), плоские участки, седловые точки и длинные узкие овраги.

Аналитическими средствами будет невозможно определить положение глобального минимума на поверхности ошибок, из-за этого обучение нейронной сети сводится к исследованию поверхности ошибок. Отталкиваясь от случайной начальной конфигурации весов и порогов (т.е. случайно взятой точки на поверхности ошибок), алгоритм обучения постепенно находит глобальный минимум. Для этого вычисляется градиент (наклон) поверхности ошибок в данной точке, и позже эта информация используется для продвижения вниз по склону. По итогу алгоритм делает остановку в нижней точке, которая может являться лишь локальным минимумом (глобальным минимумом).

Глава 2. Классические алгоритмы машинного обучения

В данной главе будут рассмотрены классические алгоритмы машинного обучения (МО) как класс методов автоматического создания прогнозных моделей на основе данных. Какой алгоритм работает лучше всего (контролируемый, неконтролируемый, классификация, регрессия и т.д.), зависит от типа решаемой задачи, доступных вычислительных ресурсов и характера данных.

2.1. Регуляризация

Регуляризация - считается эффективным методом устранения переобучения. Ведь истинное распределение данных неизвестно и эмпирический риск, взятый на основе эмпирического распределения, будет иметь склонность к переобучению. Поэтому, лучшей стратегией будет являться ситуация, где есть возможность хорошо подогнать данные обучения, а после применить метод регуляризации, для хорошего обобщения модели.

Одним из неявных допущений методов регуляризации, таких как регуляризация параметров L2 и L1, заключается в том, что величина параметров должна быть равна нулю и стремиться сжать все параметры до нуля. Это говорит, что необходимо избегать очень хорошего следования обучающим данным, что может заставлять алгоритм обучения выделять некоторый шум, бесполезный при использовании к невидимым данным.

- Значение λ должно быть настроено для получения наилучшей ошибки обобщения. Обычно применяется набор проверки при сравнении моделей со значениями для λ_s и выбирается модель с наименьшей ошибкой проверки.
- Применять регуляризацию только тогда уместно, когда модель страдает от переобучения.

- Если после использования регуляризации ошибка валидации так же велика, то, можем сказать, что находимся в регионе недостаточного соответствия. Т.е. наша модель всё ещё слишком проста и уже имеет высокий уклон. Поэтому стоит добавить сложности в модель, а затем использовать регуляризацию.

L1-регуляризация

В основе L1 лежит добавление штраф к изначальной функции затрат, в L1-регуляризации применяется специальное L1-нормирование. Называется эта регуляризация лассо-регрессией (Lasso).

$$J_{LASSO} = \sum_{\mathrm{i=1}}^{\mathrm{N}} (\mathrm{y_n} - \hat{y}_n)^2 + \lambda \|w\|_1 ,$$

L2-регуляризация

Называется ещё регрессией Риджа (Ridge). Суть состоит в том, что проводится изменение первоначальной функции, добавлением «штрафа» на большие весовые коэффициенты. Для этого добавляем постоянную, умноженную на квадрат *w*:

$$J_{RIDGE} = \sum_{\mathrm{i=1}}^{\mathrm{N}} (\mathrm{y_n} - \hat{y}_n)^2 + \lambda \|w\|_2^2 \quad ,$$

$$|w|^2 = w^T w = w_1^2 + w_2^2 + \ldots + w_D^2.$$

Отличия между L1 и L2-регуляризациями

Известно, что L1-регуляризация содействует разреженности функции, когда лишь немногие факторы не равны нулю. L2-регуляризация благоприятствует появлению малых весовых коэффициентов модели, но не способствует их точному равенству нулю. Потому что нужно обратить внимание на то, что оба

метода помогают улучшить обобщение и ошибки теста, так как не допускают переобучения модели из-за шума в данных.

- L1-регуляризация реализует это путём отбора наиболее важных факторов, сильнее всех влияющие на результат. Можно считать, что факторы с малой величиной влияния на конечный результат фактически «помогают» вам предсказывать только шум в наборе обучающих данных.
- L2-регуляризация препятствует переобученности модели путём запрета на непропорционально огромные весовые коэффициенты.
- Контурные графики отрицательных логарифмов на рис.6 показывают разницу между каждым из типов регуляризации.

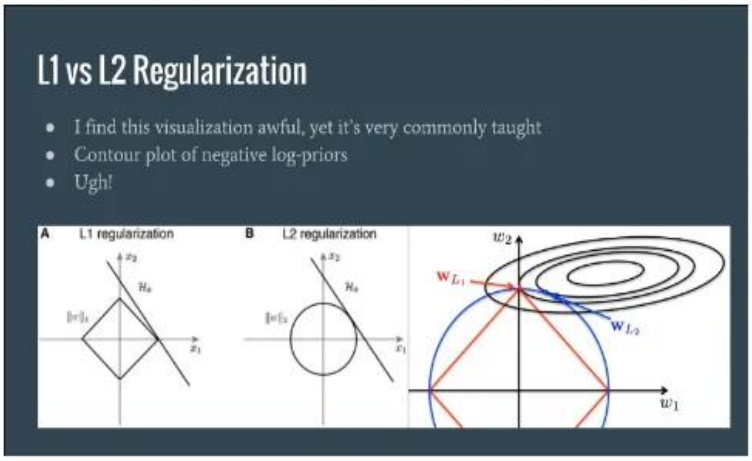

Рис.6.Отличия L1 и L2 регуляризации

При L2-регуляризации дополнительный член будет являться квадратичной функцией, при L1-регуляризации – модулем. Здесь производная функции, является ключевой, ведь градиентный спуск в основном движется в направлении производной. При квадратичном члене чем ближе нахождение к нулю, тем меньшей будет производная, пока также не приблизится к нулю. Поэтому при L2-регуляризации, когда величина *w* уже мала, дальнейший градиентный спуск уже её сильно не изменит. В случае модуля производная будет являться константой с абсолютной величиной, равной единице. Формально в нуле она не определена, но считаем её также равной нулю.

Поэтому при L1-регуляризации градиентный спуск будет стремиться к нулю с постоянной скоростью, а достигнув его, там и останется. Из-за этого L2-регуляризация способствует малой величине весовых коэффициентов, а L1-регуляризация способствует их равенству нулю, тем самым провоцируя разрежённость.

Эластичная сеть

Это модель регрессии с двумя регуляризаторами L1, L2. Вот такая модель даже имеет специальное название – *ElasticNet*. Просто добавление и штрафа L1-регуляризации, и штрафа L2-регуляризации к функции затрат. Пусть известны измерения n объектов. Каждый объект представим в виде пары (x_i, y_i), $x_i \in R^k$, $y_i \in R$. Для удобства запишем это в матричном виде: (X, y). Классическая задача регрессии ставится следующим образом:

$$|y - Xw|^2 \rightarrow min_w$$

В силу неточности измерений данных или каких-либо ещё ошибок с целью построения наилучшей модели вводят регуляризатор или несколько регуляризаторов. Тогда получаем такую задачу оптимизации:

$$|y - Xw|_2^2 + \lambda_1|w|_1 + \lambda_2|w|_2^2 \rightarrow min_w$$

или

$$J_{ELASTICNET} = J + \lambda_1|w| + \lambda_2|w|^2.$$

2.2. Линейная регрессия

Линейная регрессия (англ. Linear regression) — используемая в статистике

регрессионная модель зависимости одной (объясняемой, зависимой) переменной от другой независимой.

Линейная регрессия относится к контролируемому обучению. Есть ряд значений данных x и y. Нанеся их на график, можно затем рассматривать каждое значение x как входную переменную, в результате которой на выходе получать некоторое исходящее значение y. Контролируемое обучение, в свою очередь, также делится на две большие категории – классификацию и регрессию.

- Классификация пытается определить категорию входных данных, наличие или отсутствие их какой-то особенности – например, пытается распознать написанную цифру или определить, содержится ли на изображении кот
- Регрессия же вычисляет определенное число или вектор – например, завтрашнюю температуру или в нашем случае цены на акции BMW.

Рассмотрим теперь собственно линейную регрессию. Здесь имеется зависимость между x и y, причём исходящее значение y является числом. Многие регрессию называют также «линией наилучшего соответствия», так как если нанести на график точки данных x и y и провести соответствующим образом прямую, то получится линия наилучшего соответствия.

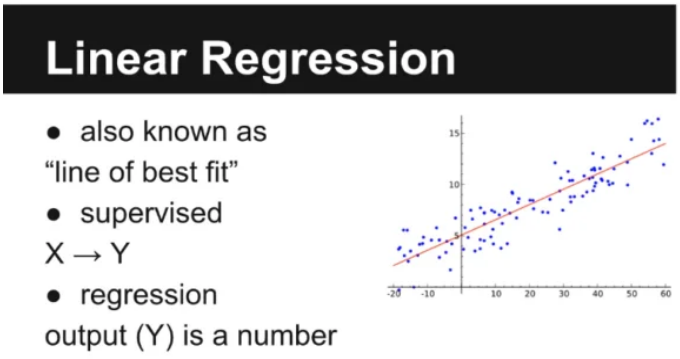

Рис.7.График линейной регрессии

Целевая функция линейной регрессионной модели:

$$y = f(x, b) + \varepsilon, E(\varepsilon) = 0$$

имеет вид:

$$f(x, b) = b_0 + b_1 x_1 + \ldots + b_k x_k$$

где b_i - параметры (коэффициенты) регрессии, x_i - регрессоры (факторы модели), k — количество факторов модели.

Многомерная линейная регрессия

Линейная регрессия использует простой функционал качества - среднеквадратичную ошибку. В ходе задачи нам будет предстоять работа с выборкой, содержащей 6 признаков. Для настройки параметров (весов) модели решается следующая задача:

$$\frac{1}{l}\sum_{i=1}^{l}((w_0 + w_1 x_{i1} + w_2 x_{i2} + \ldots + w_7 x_{i6}) - y_i)^2 \rightarrow \min_{w_1, w_2 \ldots w_6},$$

где $x_{i1}, x_{i2} \ldots x_{i6}$ - значения признаков i-го объекта, y_i - значение целевого признака i-го объекта, ℓ - число объектов в обучающей выборке.

Многомерная линейная регрессия, называется также множественной линейной регрессией.

Так выглядит решение уравнения множественной линейной регрессии. В библиотеках Python есть специальные функции для решения уравнений вида Ax=b. При программировании алгоритма обозначим так:

$$A = (X^T X),\ x = w \text{ и } b = X^T y.$$

2.3. Случайный лес

Композиции решающих деревьев часто показывают высокое качество работы во многих задачах. Популярным и эффективным методом является случайный лес (Random Forest). Построение каждого решающего дерева может учитывать и категориальные признаки. Для этого в каждом узле для выбора разбиения происходит перебор разбиений значений признаков на два множества, т.е. на левого и правого потомка в дереве.

Поскольку перебор всевозможных разбиений значений признаков является очень вычислительно затратной задачей и имеет вычислительную сложность порядка $O(2q)$, где q- количество принимаемых признаком значений, то на практике чаще используется вариант с перебором некоторой произвольной части разбиений.

Случайный лес (Random forest) представим в виде ансамбля деревьев решений, которые обучаются независимо. Каждый из решающих деревьев, обычно, делают простым. Для принятия окончательного решения выбирается значение, за которое проголосовало большинство деревьев. Каждое дерево в отдельности даёт низкое качество, но за счет их большого числа результат улучшается [1]. На рис.8 продемонстрирован пример случайного леса из двух деревьев.

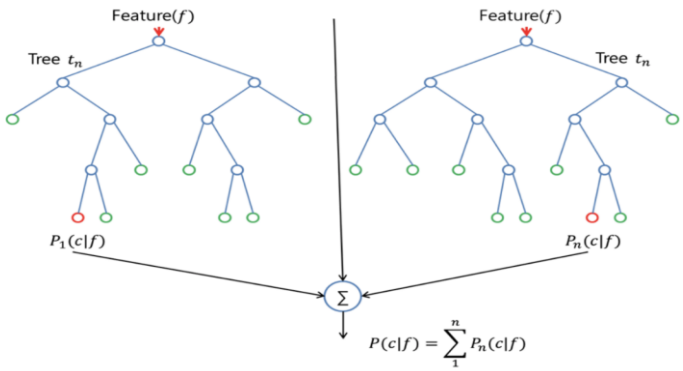

Рис.8 – Случайный лес из 2 деревьев

В построении деревьев на данных существуют свои особенности. Опишем традиционный подход в построении дерева.

Пусть N - количество примеров на обучении. Тогда:

1. выбирается подвыборка обучающей выборки размера N (возможно, с возвращением)
2. строится дерево. При поиске разбиения при построении каждого узла рассматриваются не все признаки, а только часть из них (sklearn реализация - корень из количества признаков)
3. дерево строится до полного исчерпания подвыборки, либо на основе каких-либо других критериев

Каждое из деревьев стараются сделать как можно проще. При принятии решения выбирается класс, за который проголосовало большинство деревьев решений.

Случайный лес дает сравнительно плохие результаты при разделении выборки в метрическом пространстве. В том случае, когда ожидается некоторый «правильный» вид разделяющей гиперплоскости. Тем не менее, у случайного леса есть большое число достоинств, которые могут использоваться в рассматриваемой задаче определения вредоносности файла.

Алгоритм не оперирует метриками, что позволяет свободно работать с признаками разной природы. Возможно обрабатывать данные с пропущенными значениями признаков. Высокая скорость работы алгоритма, независимое построение решающих деревьев.

2.4. Метод *k* ближайших соседей

Также в качестве сравниваемого решения был выбран метод k ближайших соседей (kNN). Алгоритм kNN требует способа вычисления расстояния между объектами – метрики [1], то есть Основой метода является гипотеза компактности: если метрика расстояния между примерами введена весьма удачно, то похожие примеры намного чаще находятся в одном классе, чем в разных. Является популярным методом классификации, часто используется в задачах регрессии. Считается, одним из самых понятных подходов к классификации.

Пусть задана обучающая выборка пар «объект-ответ»

$$X^m = \{(x_1, y_1), \dots, (x_m, y_m)\}$$

Пусть на множестве объектов задана функция расстояния

$$p(x, x')$$

Эта функция должна быть достаточно адекватной *моделью сходства* объектов. Чем больше значение этой функции, тем менее схожими являются два объекта $x, x`$.

Для произвольного объекта u расположим объекты обучающей выборки x_i в порядке возрастания расстояний до u

$$p(u, x_{1;u}) \leq p(u, x_{2;u}) \leq \cdots \leq p(u, x_{m;u})$$

где через $x_{i;u}$обозначается тот объект обучающей выборки, который является $i-$м соседом объекта u

В наиболее общем виде алгоритм KNN выглядит так:

$$a(u) = argmax_{y \epsilon Y} \sum_{i=1}^{m} [y(\mathrm{x_{i;u}}) = y] w(i,u)$$

2.5. Функции потерь

В многих обучающих сетях ошибка рассчитывается как разница между фактическим выходным значением y и прогнозируемым выходным значением $\hat{y}$. Функция, применяемая для вычисления этой ошибки - функция потерь или функция затрат, ошибки.

Средняя квадратичная ошибка (MSE): является одной из самых распространенных функцией потерь. Функция потерь MSE массово применяется в линейной регрессии в качестве показателя эффективности. Чтобы рассчитать MSE, нужно взять разницу между предсказанными значениями и истинными, возвести её в квадрат и усреднить по всему набору данных.

$$\mathrm{MSE} = \frac{1}{m} \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||^2$$

где $y^{(i)}$ – фактический ожидаемый результат, а $\hat{y}^{(i)}$ – прогноз модели.

Функция детерминации (R^2): Нам необходим какой-то количественный показатель того, насколько хороша наша модель. Обычно для этого в линейной регрессии – и не только в линейной, но и других – используется коэффициент детерминации R^2. Определяется как:

$$R^2 = 1 - \frac{SSres}{SStot}$$

где $SSres$ сумма квадратов остатков регрессии, $SStot$ — общая сумма квадратов ошибки.

2.6. Обзор библиотек и фреймворков использованных в работе

Большинство представленных фреймворков представляют модель в виде вычислительного графа, статического или динамического.

Фреймворки, объявляющие графы вычислений статически, такие как TensorFlow, позволяют сначала определить архитектуру графа, а затем выполнить его много раз, прогнав через него некоторый объём данных. Это позволяет легко распределять задачи для выполнения на разных машинах. В дополнение к этому, фреймворки могут оптимизировать граф для нас заранее. Они также позволяют пользователям сериализовать график после его создания и выполнять его, не требуя кода, который его построил.

Tensorflow

Для реализации выбранных моделей, выбор пал на фреймворк TensorFlow. Популярность данного фреймворка вполне заслужена: имеется поддержка большого числа разработчиков, обширный выбор документации, открытый исходный код, распределённое выполнение наших моделей, различные архитектурные оптимизации, визуализацию во время выполнения (что упрощает отладку модели) и портативность.

Keras

В качестве фреймворка для имплементации возьмем *Keras* — он очень прост, интуитивно понятен и имеется возможность просто реализовывать достаточно сложные вычислительные графы. Также Keras позволяет достаточно гибко контролировать процесс обучения [7].

Jupyter Notebook

Возможности данного фреймворка позволяют взаимодействовать со средой

Python через интерактивный веб-интерфейс путем ввода команд в поля веб-ноутбука (рис.9). Пакет Jupyter Notebook используется в совокупности с большим количеством библиотек и дополнений.

Одним из самых важных из них является пакет Pandas. Данная библиотека поддерживает работу как со встроенными типами Python, так и загрузку данных с диска в таких форматах, как TXT, TSV, CSV и Microsoft Excel. Также пакет предоставляет базовые возможности по предобработке и визуализации данных.

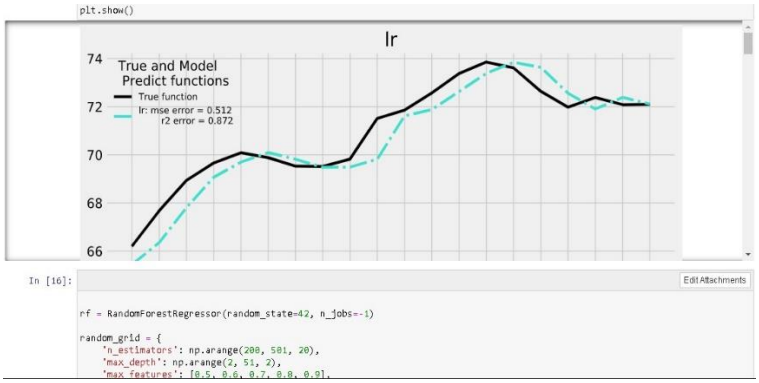

Рис 9. Окно взаимодействия с интерактивным редактором Jupyter Notebook

Следующим пакетом, без которого невозможно какое-либо исследование в данной среде является пакет NumPy. Основным её преимуществом перед библиотекой Pandas является широкий набор функционала по работе с многомерными данными, без которых невозможно представить ни одну комплексную модель машинного обучения.

Также необходимо отметить пакет Scikit-learn, содержащий в себе набор ряда алгоритмов машинного обучения, используемых для создания моделей. Данный пакет содержит в себе обширный набор методов для предобработки, анализа и визуализации данных. Эта библиотека включает в себя такие методы классификации, как DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier и другие.

ГЛАВА 3. Подготовка данных и построение предсказательной модели. Обучение

В этой главе проведём разбор построения моделей RNN и введем несколько важных определений используемых при решении задач признаков. Построим модели LSTM и MLP настроим их и обучим. Произведём настройку гиперпараметров алгоритма «Случайный лес».

3.1. Набор данных. Методика прогноза. OHLC

Набор данных можно загрузить с Yahoo! Finance. В данной работе использовались данные акций компании «BMW» с **01.01.2003** г. до **01.01.2019** г. В работе в качестве набора данных будем использовать ежедневные цены закрытия, а также позже другие важные признаки, такие как Volume и OHLC. Прогноз алгоритма на определённый день будет осуществляться за счёт предыдущих 10 дней торгов и т.д. в течении одного выбранного нами месяца. В результате работы получится прогноз на 30 дней вперёд.

Индикатор volume:

Осветим основные закономерности использования индикатора volume. При однонаправленном росте объёма и цен, следует ситуация, что рынок поддерживает тренд. В этом случае при восходящем движении цен советуют покупать, при нисходящем – продавать. Если же объем снижается при росте цен, это означает отсутствие поддержки тренда со стороны рынка. Такая ситуация сигнализирует о возможном развороте. **Volume** часто предопределяет изменения цен. Когда цена растет вместе с объёмами, а после начинает снижение, некоторое время цена будет еще расти по инерции.

Главным недостатком индикатора является его подверженность сильным скачкам. Его значения могут искажать реальную ситуацию, особенно в случае использования на валютном рынке. Поэтому нужно предварительно тестировать его возможности.

OHLC

Это сокращенное обозначение котировок, которые указываются для элементарной диаграммы какого-либо ценового графика. OHLC бар является важным этапом в торговле без индикаторов [6].

В этой аббревиатуре:

O обозначает **Open** – цену открытия интервала,

H означает **High** — максимум цены интервала,

L означает **Low** – минимум цены интервала,

C означает **Close** – цену закрытия интервала.

Каждый график типа OHLC (рис.10), представляет собой классический столбиковый график, бар. В нем каждый интервал времени (например, 5 минут) представлен OHLC ценами прямо внутри этого интервала. Другими словами, это основная единица торговли на бирже, содержащая цены открытия и закрытия, максимальное и минимальное значение цены за период времени, содержащийся в баре.

Рис.10.Визуальное представление признаков

Цена открытия — **Open**, означает нахождение цены в момент самого начала данного бара, то есть в самый начальный момент времени определённого временного отрезка, описанного в баре. Этот параметр

указывает с какой цены начался определенный временной отрезок на рынке акций.

Цена закрытия— **Close**, значением, которое приобретает цена за определённый промежуток времени, содержащийся в конкретном баре. Нужно сказать, что цена закрытия может не совпадать с ценой открытия следующего бара, особенно тогда, когда цены меняются очень быстро (бывают и такие моменты). Между ценой закрытия старого бара и открытия нового проходит достаточно много времени, которого хватает для изменения цены.

Максимальное и минимальное значение цены — **High** и **Low**, говорят о наибольшем или наименьшем значении, которое приобреталось ценой за время формирования одного конкретного бара.

3.2. Подготовка данных

Рекуррентная нейронная сеть (RNN) представляет собой тип искусственной нейронной сети с рекуррентными переходами в скрытых слоях, что позволяет ей использовать предыдущее состояние скрытых нейронов для получения нового результата с учётом нового ввода. RNN умеет обрабатывать последовательные данные.

Долгая краткосрочная память (LSTM) - специально разработанная ячейка, помогающая RNN лучше запоминать долгосрочный контекст. Цены на акции представимы в виде временных рядов длины N, определённые как $p_0, p_1, ..., p_{N-1}$, в которых p_i - цена закрытия в день i, $0 \le i < N$.

Пусть имеется скользящее окно фиксированного размера w называемый input_size, перемещаем его вправо на w, чтобы не было перекрытия между данными во всех скользящих окнах.

Модель RNN, которую мы строим, имеет LSTM-ячейки в качестве основных скрытых элементов. Мы используем значения с самого начала обучения в первом скользящем окне W_0 до окна W_t в момент времени t:

$$W_0 = (p_0, p_1, \ldots, p_{w-1})$$
$$W_1 = (p_w, p_{w+1}, \ldots, p_{2w-1})$$
$$\ldots$$
$$W_t = (p_{tw}, p_{tw+1}, \ldots, p_{(t+1)w-1})$$

для прогнозирования цен в следующем окне W_{t+1}:

$$W_{t+1} = (p_{(t+1)w}, p_{(t+1)w+1}, \ldots, p_{(t+2)w-1})$$

или

$$f(W_0, W_1, \ldots, W_t) \approx W_t + 1.$$

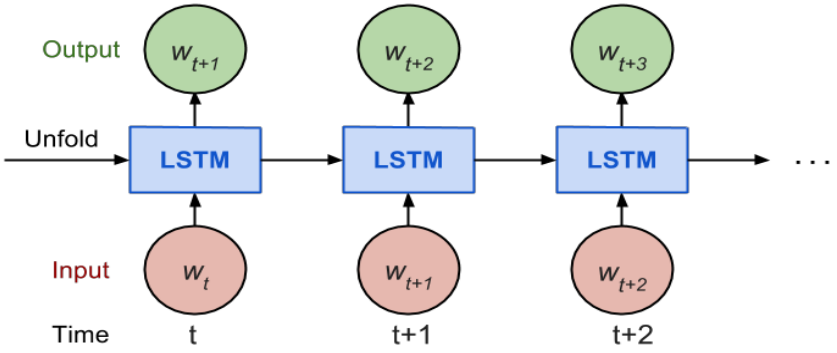

Рис. 11. Развернутая версия RNN.

Рассматривая, как работает **обратное распространение во времени (Backpropagation through time, BPTT)** мы обычно обучаем RNN в «развёрнутой» версии, так что не имеет смысла проводить обратное распространение очень далекое по времени, увеличивая сложность обучения. Вот объяснение по num_steps из документации Tensorflow:

Выход рекуррентной нейронной сети (RNN) зависит от произвольно отдалённых входов. К сожалению, это затрудняет вычисления обратного распространения. Для того, чтобы сделать процесс обучения приемлемым, обычной практикой является создание «развёрнутой» версии сети, которая

содержит фиксированное число (num_steps) входов и выходов LSTM. Затем модель обучается в этом конечном приближении RNN.

Это может быть реализовано путем подачи входных данных длиной *num_steps* за один раз и выполнения обратного прохода после каждого такого входного блока [2]. Последовательность цен сначала разделяется на небольшие не перекрывающие друг друга окна. Каждое из них содержит *input_size* числа, каждое из которых рассматривается как один независимый элемент ввода. Затем любые последовательные элементы ввода *num_steps* группируются в один ввод для обучения, образуя «**нераскрученную**» версию RNN для обучения в TensorFlow. Соответствующая метка является элементом ввода сразу после них.

В рекуррентной нейронной сети на одном временном шаге ***t***, входной вектор содержит input_size (помеченные как ***W***) суточные значения цены ***i***-й акции, $(p_i, t_w, p_i, t_{w+1}, ..., p_{i,(t+1)w-1})$. Акция однозначно отображается на вектор длины embedding_size (помечен как ***k***), $(e_i, 0, e_i, 1, ..., e_i, k)$. Как показано на рис. 14, вектор цены конкатенируется с вектором вложения, а затем подаётся в ячейку LSTM.

Другой альтернативой является конкатенация векторов вложения с последним состоянием ячейки LSTM и получением новых весов ***W*** и смещение ***b*** в выходном слое. Но, таким образом, LSTM-элемент не может отличить цены одной акции от другой, и его мощность будет в значительной степени сдерживаться.

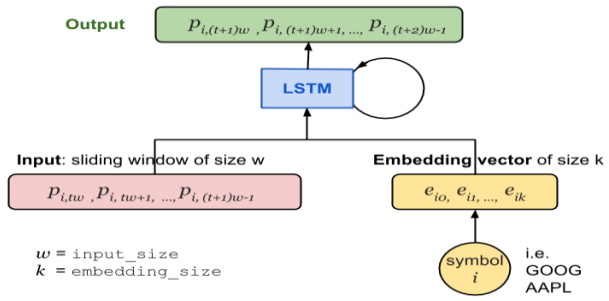

Рис.12. Архитектура модели RNN для прогнозирования цен на акции с вложением.

3.3. Нормализация данных

Существует несколько различных методов нормализации данных, в данной работе будет использоваться такой метод:

Нормализация Min-max устанавливает наименьшим наблюдаемым значением - 0, а наибольшим наблюдаемым значением — 1.

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Нормализация на стандартное отклонение.

$$z = \frac{x - \mu}{\sigma}$$

$$\mu - \text{mean}, \sigma - \text{standart deviation}$$

Метод главных компонент (PCA)

Анализ основных компонентов - статистическая процедура для преобразования набора наблюдений за возможными коррелированными переменными в набор значений линейно некоррелированных переменных.

Каждый из основных компонентов выбирается таким образом, чтобы он описывал большую часть все еще доступной дисперсии, и все эти основные компоненты ортогональны друг другу. Во всех основных компонентах первый главный компонент имеет максимальную дисперсию.

Использование PCA:

- для поиска взаимосвязи между переменными в данных.
- для интерпретации и визуализации данных.
- Так как количество переменных уменьшается, это упрощает дальнейший анализ.

3.4. Построение и настройка модели

Для начала подключим и импортируем нужные для работы библиотеки и скрипты в Python:

```
import tensorflow as tf
import pandas_datareader.data as pdr
import yfinance as yf
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split, cross_val_score, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.decomposition import PCA
```

Настройка нейронной сети LSTM

Определения, которые будут использоваться в модели:

start и end - период, на котором будет проходить обучение. **start_test** и **end_test -** период, на котором будет проходить предсказание. **seq_len** - количество предыдущих дневных цен (признаков), на основе которых делается предсказание цены дня.

```
start = "2003-01-01"
end = "2019-04-01"
#start_test = "2017-12-15"
#end_test = "2018-02-01"
start_test = "2019-03-19"
end_test = "2019-05-01"
ticker = "BMW.DE"
train_file_all = "stock_prices_all.csv"
test_file_all = "test_all.csv"
scal_file_all = "scal_all.csv"
seq_len = 10
```

Получает исторические данные по дневным ценам акций между датами:

```
def get_stock_data_all(ticker, start_date, end_date, file):
```

param **ticker** - компания или компании, чьи данные должны быть извлечены

type **ticker** - string or list of strings

param **start_date** - начальная дата получения цен на акции

type **start_date** - string of date "YYYY-mm-dd"

param **end_date** - конечная дата получения цен на акции

type **end_date** - string of date "YYYY-mm-dd"

param **file** - имя возвращаемого файла с данными

return - файл формата csv.

Преобразуем данные, разбивая на признаки и ответы:

```
def get_X_Y_all(data, seq_len, list_x, list_x_all, list_x_1, list_x_2, list_x_3, list_x_4, list_x_5, list_y):
    """
```

param **data** - исходный массив данных

param **list_x** - список, в который добавляются признаки

param **list_y** - список, в который добавляются ответы

Получим три массива данных - тренировочный, для предсказания и для настройки скалера:

```
get_stock_data_all(ticker, start, end, train_file_all)
get_stock_data_all(ticker, start_test, end_test, test_file_all)
get_stock_data_all(ticker, start, end_test, scal_file_all)
```

Разобьем полученные данные на признаки и ответы.

```
data = pd.read_csv(train_file_all, encoding='utf-8')
get_X_Y_all(data, seq_len, x, x_all, x_1, x_2, x_3, x_4, x_5, y)

new_data = pd.read_csv(test_file_all, encoding='utf-8')
get_X_Y_all(new_data, seq_len, new_x, new_x_all, new_x_1, new_x_2, new_x_3, new_x_4, new_x_5, new_y)

scal_data = pd.read_csv(scal_file_all, encoding='utf-8')
get_X_Y_all(scal_data, seq_len, scal_x, scal_x_all, scal_x_1, scal_x_2, scal_x_3, scal_x_4, scal_x_5, scal_y)
```

Преобразуем данные в формат, используемый в нейронной сети:

```
x = np.array(x)
x_all = np.array(x_all)
x_1 = np.array(x_1)
x_2 = np.array(x_2)
x_3 = np.array(x_3)
x_4 = np.array(x_4)
x_5 = np.array(x_5)
y = np.array(y)
new_x = np.array(new_x)
new_x_all = np.array(new_x_all)
new_x_1 = np.array(new_x_1)
new_x_2 = np.array(new_x_2)
new_x_3 = np.array(new_x_3)
new_x_4 = np.array(new_x_4)
new_x_5 = np.array(new_x_5)
new_y = np.array(new_y)
scal_x = np.array(scal_x)
scal_x_all = np.array(scal_x_all)
scal_x_1 = np.array(scal_x_1)
scal_x_2 = np.array(scal_x_2)
scal_x_3 = np.array(scal_x_3)
scal_x_4 = np.array(scal_x_4)
scal_x_5 = np.array(scal_x_5)
scal_y = np.array(scal_y)
```

Произведём скалинг данных:

```
X = [x_all, x_1, x_2, x_3, x_4, x_5]
NEW_X = [new_x_all, new_x_1, new_x_2, new_x_3, new_x_4, new_x_5]

scalers = {}
```

```
scaler_y = MinMaxScaler(feature_range = (0, 1))
scaler_y.fit_transform(scal_y)
y = scaler_y.transform(y)
```

Для признаков OHLC проведём дополнительную нормализацию данных с помощью анализа Главных компонент (PCA):

```
scal_x_all_pca = scal_x_all[:, :, 1:5]
scal_x_all_pca.shape = (scal_x_all_pca.shape[0] * scal_x_all_pca.shape[1], 4)
pca = PCA(n_components = 1)
scal_XPCAreduced = pca.fit_transform(scal_x_all_pca)
```

Разобьём тренировочные данные на обучение и проверку в соотношении 9 к 1 и перемешаем их:

```
X_train, X_valid, y_train, y_valid = train_test_split(x, y, test_size=0.1, random_state=42, shuffle=True)
X_train_all, X_valid_all, y_train_all, y_valid_all = train_test_split(x_all, y, test_size=0.1, random_state=42, shuffle=True)
X_train_1, X_valid_1, y_train_1, y_valid_1 = train_test_split(x_1, y, test_size=0.1, random_state=42, shuffle=True)
X_train_2, X_valid_2, y_train_2, y_valid_2 = train_test_split(x_2, y, test_size=0.1, random_state=42, shuffle=True)
X_train_3, X_valid_3, y_train_3, y_valid_3 = train_test_split(x_3, y, test_size=0.1, random_state=42, shuffle=True)
X_train_4, X_valid_4, y_train_4, y_valid_4 = train_test_split(x_4, y, test_size=0.1, random_state=42, shuffle=True)
X_train_5, X_valid_5, y_train_5, y_valid_5 = train_test_split(x_5, y, test_size=0.1, random_state=42, shuffle=True)
X_train_6, X_valid_6, y_train_6, y_valid_6 = train_test_split(x_6, y, test_size=0.1, random_state=42, shuffle=True)
X_train_pca, X_valid_pca, y_train_pca, y_valid_pca = train_test_split(x_pca, y, test_size=0.1, random_state=42, shuffle=True)
X_train_all_pca, X_valid_all_pca, y_train_all_pca, y_valid_all_pca = train_test_split(x_all_pca, y, test_size=0.1, random_state=42, shuffle=True)
```

Модель *Sequential* представляет собой линейный стек слоёв. Создадим модель *Sequential*, передав в конструктор список экземпляров слоя. Для первых двух слоев используем LSTM - рекуррентную нейронную сеть "Долгая краткосрочная память".

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(20, input_shape=(10, 5), return_sequences=True))
model.add(tf.keras.layers.LSTM(20))
model.add(tf.keras.layers.Dense(1, activation=tf.nn.relu))
```

В этой модели мы укладываем 2 слоя LSTM друг на друга, что делает модель способной к обучению темпоральных представлений более высокого уровня.

Первая LSTM возвращает свою полную выходную последовательность, но вторая возвращает только последний шаг в своей выходной последовательности, тем самым отбрасывая временное измерение (т.е. Преобразуя входную последовательность в один вектор).

Модель должна знать, какую входную форму она должна ожидать. По этой причине первый слой в последовательной модели (и только первый, потому что следующие слои могут делать автоматический вывод формы) должен получать информацию о своей входной форме - input_shape. Укажем фиксированный размер пакета для наших входных данных (это полезно для рекуррентных сетей с сохранением состояния) - 20 образцов. Образцы в партии обрабатываются независимо, параллельно. При обучении

пакет приводит только к одному обновлению модели.

Пакет обычно аппроксимирует распределение входных данных лучше, чем один вход. Чем больше пакет, тем лучше аппроксимация; однако верно и то, что обработка пакета займет больше времени и всё равно приведет только к одному обновлению. Для вывода (оценка / прогнозирование) рекомендуется выбрать размер пакета, который настолько велик, насколько можно себе позволить, не выходя из памяти (поскольку большие пакеты обычно приводят к более быстрой оценке/прогнозированию).

Передаём слою batch_size=20 и input_shape = (10, 1), и слой будет ожидать, что каждый пакет входных данных будет иметь форму пакета (20, 10, 1) Выходной слой будет состоять из 1 нейрона с функцией активации 'relu'. Функция активации определяет выходное значение нейрона. Выходной нейрон оставим без нелинейности, чтобы иметь возможность прогнозировать любое значение.

Далее для всех 5 признаков по очереди и также нормализованных «PCA» проведём такие манипуляции:

Предскажем цены с помощью обученной модели и произведем обратный скалинг (взят пример для 2 признака):

```
y1 = scaler_y.inverse_transform(model.predict(X_valid_2))
y2 = scaler_y.inverse_transform(model.predict(new_x_2))
print(y2)
```

Запишем данные в файлы, для дальнейшего сравнения и построения графиков в другой части программы:

```
np.save('LSTM_y_pred_2', y1)
np.save('LSTM_new_y_2', y2)
```

Для MLP все действия аналогичны кроме самой настройки стека слоёв:

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(100, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(100, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(1, activation=tf.nn.relu))
```

Модель Sequential представляет собой линейный стек слоёв. Создадим модель Sequential, передав в конструктор список экземпляров слоя. Для первых двух слоёв применим MLP (multilayer perceptron). Многослойный перцептрон.

Полностью связанные слои определяются с помощью класса *Density*. Мы можем указать количество нейронов или узлов в слое в качестве первого аргумента, а также указать функцию активации, используя аргумент активации.Мы будем использовать функцию активации ReLU (rectified linear unit). Выходной слой также будет состоять из 1 нейрона с функцией активации ReLU.

Функция активации определяет выходное значение нейрона. Выходной нейрон также оставим без нелинейности, чтобы иметь возможность прогнозировать любое значение. Раньше считалось, что Sigmoid и Tanh активационные функции были предпочтительны для всех слоёв. В наши дни более высокая производительность достигается с помощью функции активации ReLU.

Проведём общие действия для двух алгоритмов:

Перед обучением модели необходимо настроить процесс обучения, что делается с помощью метода компиляции. Он получает два аргумента:

```
model.compile(optimizer="adam", loss="mean_squared_error")
```

1) Оптимизатор. Используем Adam, алгоритм градиентной оптимизации стохастических целевых функций первого порядка, основанный на адаптивных оценках моментов более низкого порядка. Этот метод прост в реализации, вычислительно эффективен, имеет небольшие требования к

памяти, инвариантен к диагональному масштабированию градиентов и хорошо подходит для задач, которые являются большими с точки зрения данных и/или параметров.

Этот метод также подходит для нестационарных задач и задач с очень шумными и / или разреженными градиентами. Эмпирические результаты показывают, что Адам хорошо работает на практике и выгодно отличается от других методов стохастической оптимизации.

2) Функция потерь. Это та цель, которую модель постарается свести к минимуму. Используем среднеквадратическую ошибку (mean squared error, MSE). Физического смысла MSE не имеет, но чем ближе к нулю, тем модель лучше.

LSTM –сеть при 100 эпохах обучается гораздо дольше чем при 50, но при этом результат практически остаётся неизменным.

```
model.fit(X_train_4, y_train_4, epochs=50)
print(model.evaluate(X_valid_4, y_valid_4))
```

А вот MLP-сеть при 100 эпохах показывает намного качественнее результат, чем при значении равном 50. Ошибка в данном случае становится минимальной и в дальнейшем уже практически не снижается.

```
model.fit(X_train_2, y_train_2, epochs=100)
print(model.evaluate(X_valid_2, y_valid_2))
```

Настройка алгоритмов машинного обучения

А именно дополнительную настройку RandomForestRegressor. Итак, воспользуемся оптимизацией гиперпараметров модели для того чтобы улучшить качество предсказаний, выдаваемых RF-моделью.

Гиперпараметры можно рассматривать как «настройки» модели. Но настройки, которые отлично подходят для одного набора данных, для другого не актуальны - поэтому и стоит заниматься их оптимизацией.

Применим алгоритм RandomizedSearchCV, позволяющий исследовать широкие диапазоны значений. В ходе работы генерируем словарь random_grid, содержащий для каждого гиперпараметра диапазон значений, которые нужно испытать. Далее, мы инициализируем объект random_search с помощью функции RandomizedSearchCV(), передавая ей RF-модель, param_dist, число итераций и число кросс-валидаций, которые нужно выполнить.

Гиперпараметр *verbose* позволяет управлять объёмом информации, который выводится моделью в ходе её работы (наподобие вывода сведений в процессе обучения модели). Гиперпараметр *n_jobs* позволяет указывать то, сколько процессорных ядер нужно использовать для обеспечения работы модели. Установка n_jobs в значение -1 приведёт к более быстрой работе модели, так как при этом будут использоваться все ядра процессора.

```
rf = RandomForestRegressor(random_state=42, n_jobs=-1)

random_grid = {
    'n_estimators': np.arange(200, 501, 20),
    'max_depth': np.arange(2, 51, 2),
    'max_features': [0.5, 0.6, 0.7, 0.8, 0.9],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10]
}

random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=random_grid,
    n_iter=50,
    scoring='r2',
    cv=10,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

random_search.fit(X_train_scaled, y_train)

print(random_search.best_score_)
print(random_search.best_params_)
```

```
Fitting 10 folds for each of 50 candidates, totalling 500 fits
```

Займёмся подбором следующих гиперпараметров: n_estimators - число «деревьев» в «случайном лесу». max_features - число признаков для выбора расщепления. max_depth - максимальная глубина деревьев. min_samples_split

- минимальное число объектов, необходимое для того, чтобы узел дерева мог бы расщепиться. min_samples_leaf - минимальное число объектов в листьях.

n_estimators	280
max_features	0.8
max_depth	8
min_samples_split	5
min_samples_leaf	1

Табл.1.Гиперпараметры для подбора в «Random Forest»

ГЛАВА 4. Визуализация и результаты

В этой главе оценим работу приложения за **апрель 2019г**, визуализируем в виде графиков качество прогнозирования нейронных сетей с добавлением, отсеиванием ранее введённых в прошлой главе дополнительных признаков в моделях, также проведём анализ полученных результатов стандартных алгоритмов машинного обучения Линейной регрессии, KNN, Случайный лес и в конце сравним все прогнозы на итоговом графике и подведём итоги работы.

4.1. Реализация

Обучим 3 различные стандартные модели алгоритмов машинного обучения и оценим точность для каждой по среднеквадратической ошибке (mean squared error, MSE) и R2- коэффициенту детерминации.

С помощью Линейной регрессии проведём предсказание. Как видим из рис.13, результат очень даже неплохой. Многие недооценивают линейную регрессию. А ведь при её относительной простоте, она выдаёт хороший прогноз.

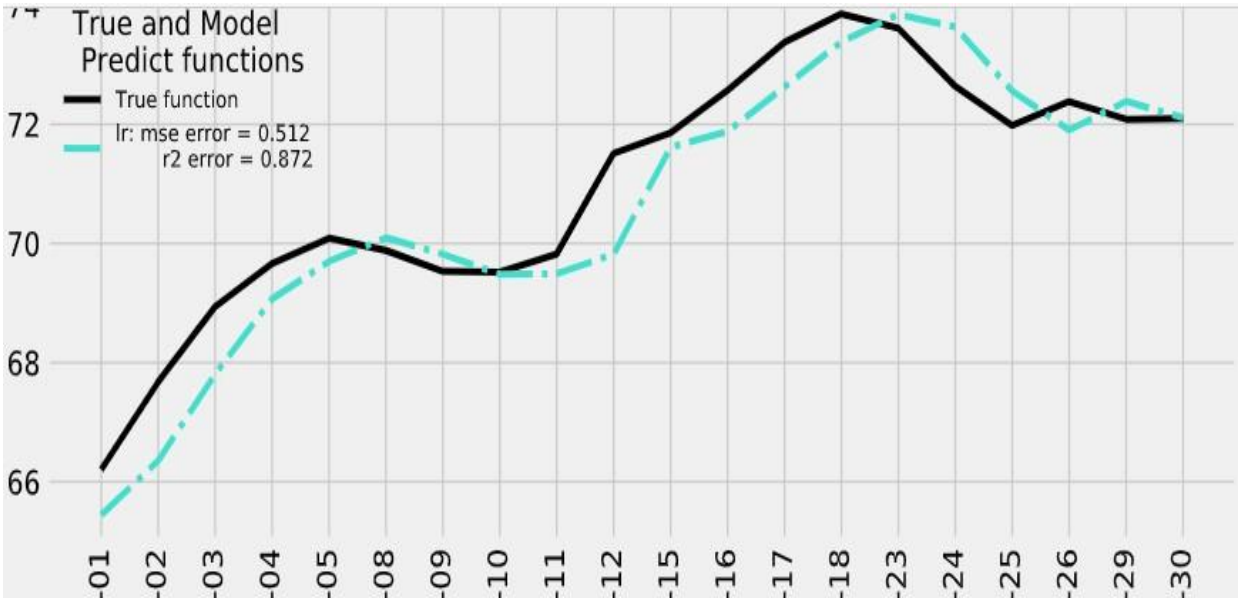

Рис.13.График работы алгоритма linear regression на периоде предсказания

На рис.14 изображена работа линейной регрессии с применением регуляризации L1:

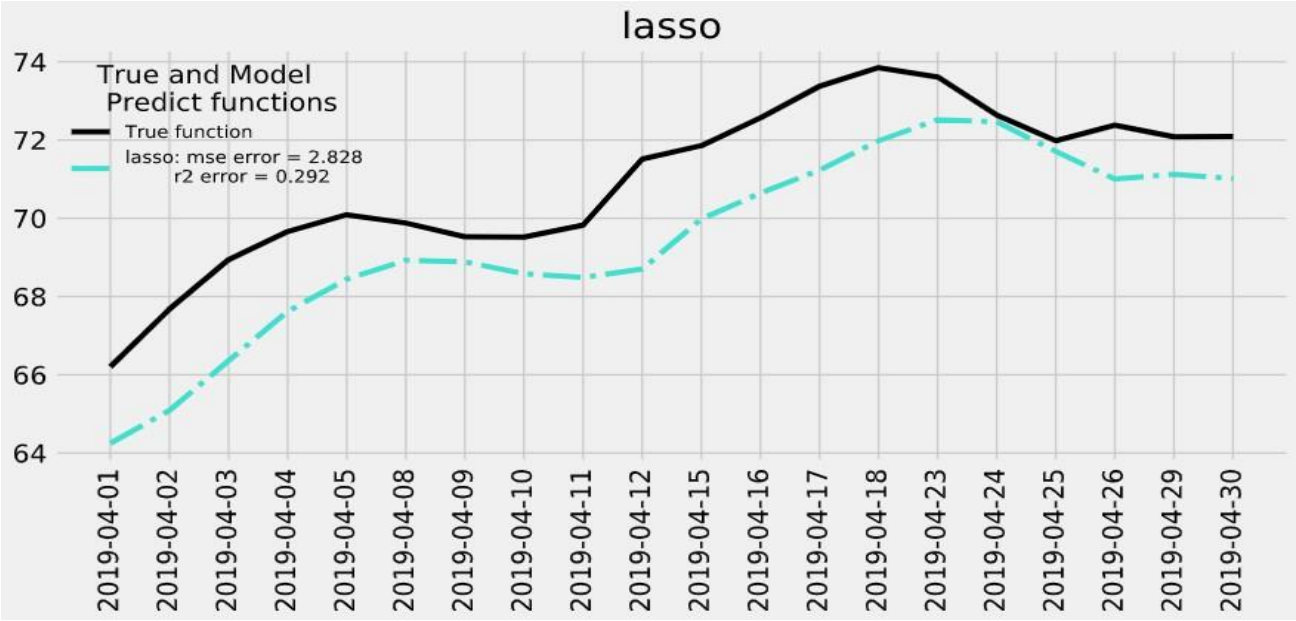

Рис.14.Применение регуляризации L1 (lasso) на периоде предсказания

На рис. 15 изображена уже работа линейной регрессии с применением регуляризации L2:

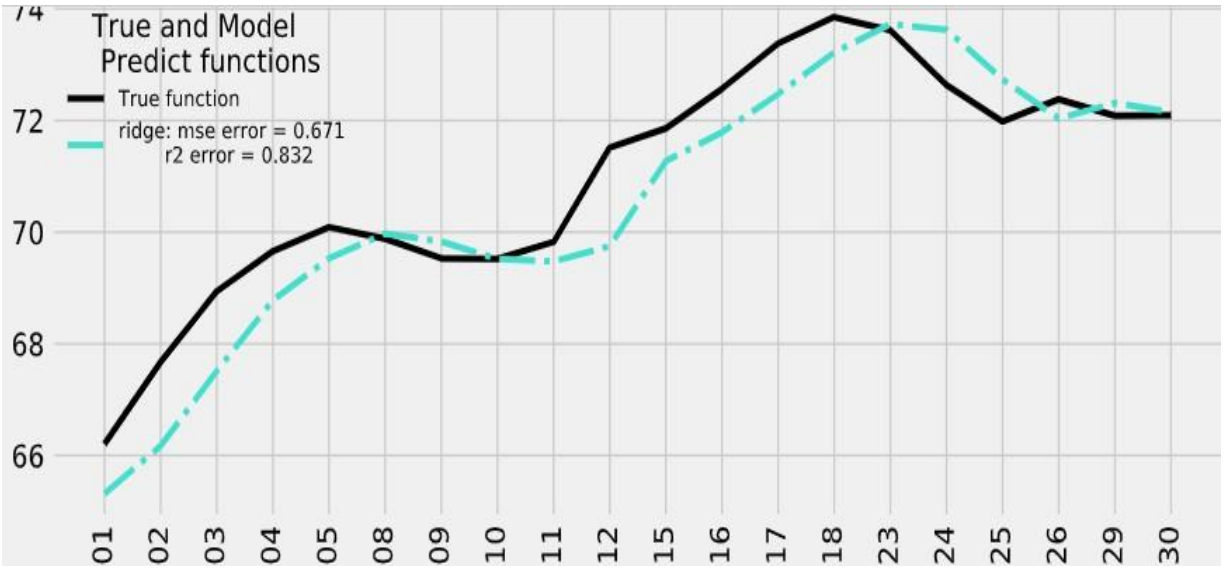

Рис.15.Применение регуляризации L2 (ridge) на периоде предсказания

На рис.16 изображена уже работа линейной регрессии с применением алгоритма суммирующего регуляризации L1 и L2, называемый ElasticNet:

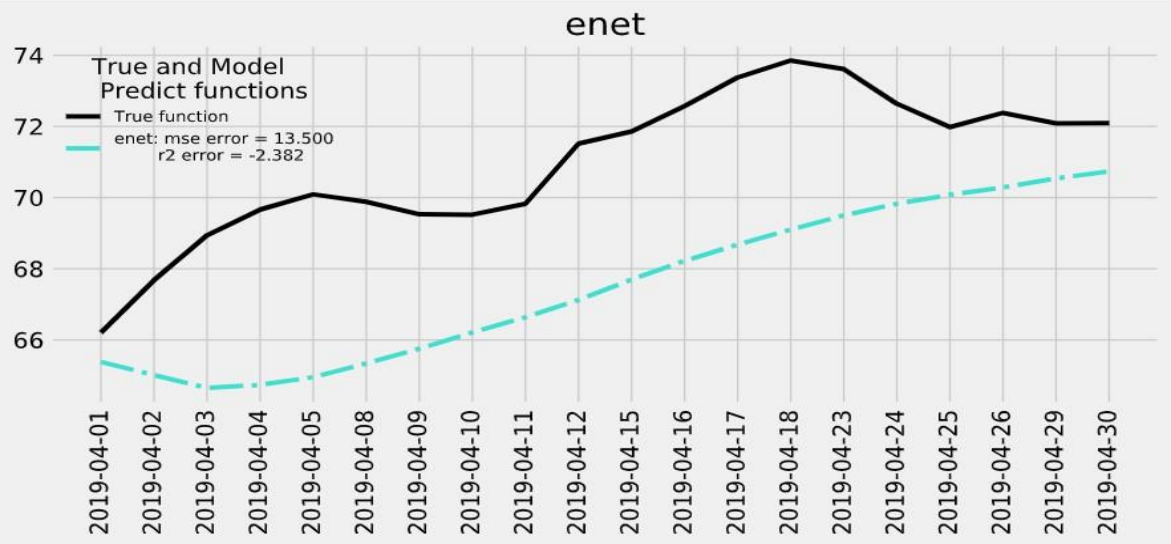

Рис.16.Применение регуляризации ElasticNet на периоде предсказания

ElasticNet сводит на нет, всю работу регуляризации Ridge, так как объединение L1 и L2 будут только запутывать модель.

Как мы видим по графикам работы алгоритма линейной регрессии с применением различных регуляризаций и без, результат прогноза будет только ухудшаться, так как данные которые берём уже оптимизированы, нет того самого шума в данных из-за которого регуляризацию стоило бы применить. Модель не страдает от переобучения.

На рис.17 изображена уже работа алгоритма KNN:

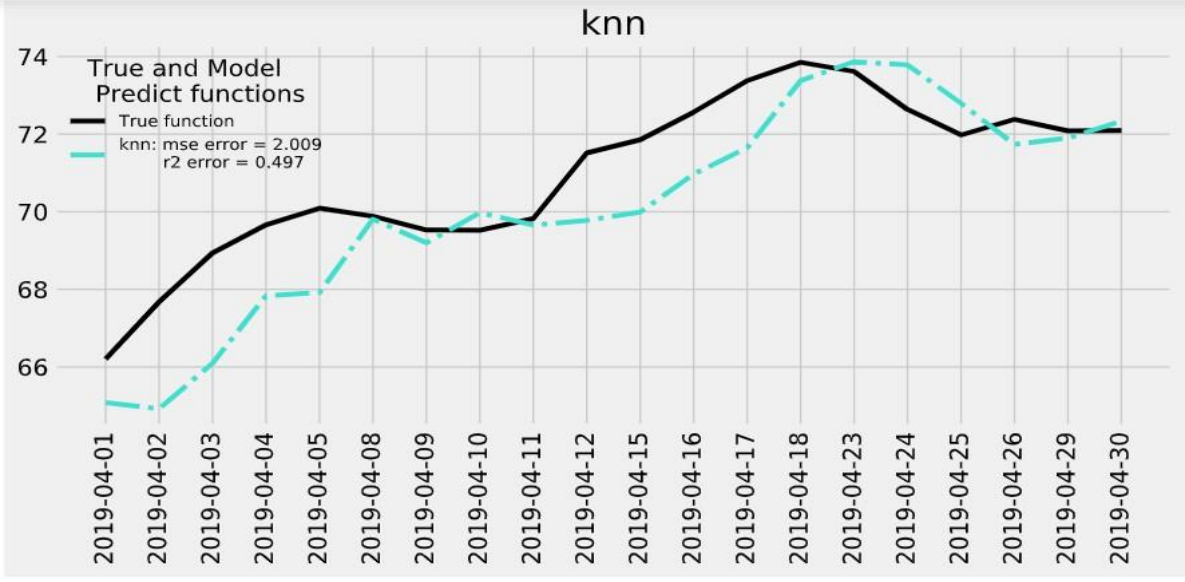

Рис.17.График работы алгоритма KNN на периоде предсказания

Как мы видим алгоритм KNN не подходит для задачи с предложенным набором данных. Выявлен большой процент ошибки, предсказания получились очень приблизительными.

На рис.18 изображена уже работа алгоритма Random forest до дополнительной настройки:

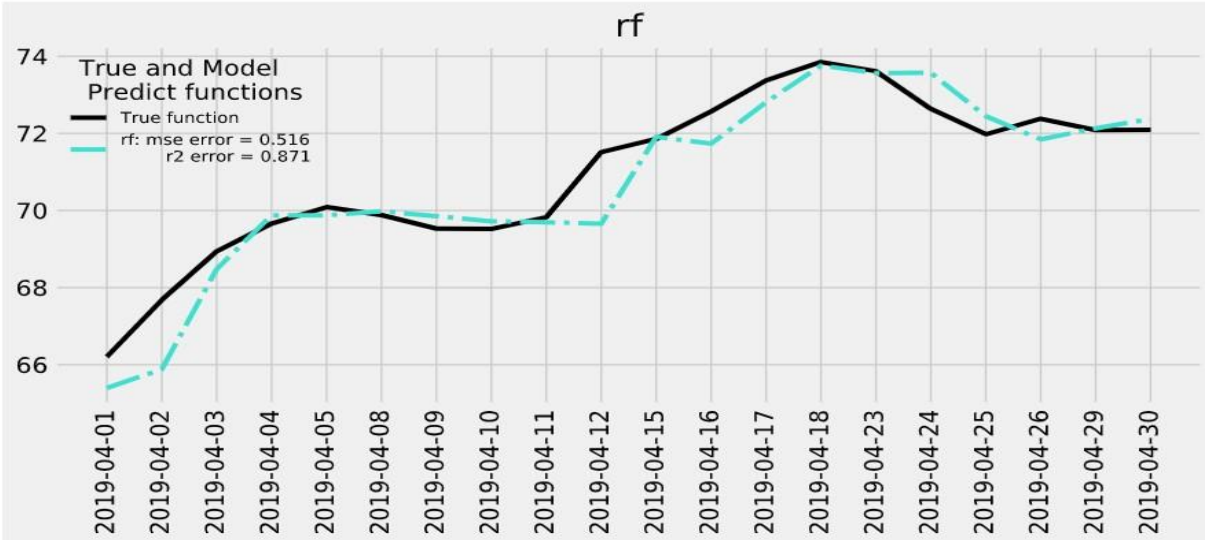

Рис.18.График работы алгоритма Random Forest на периоде предсказания

Из графика видно, что алгоритм подтвердил свою актуальность для прогнозирования таких данных. Предсказания происходят очень близко к реальной ситуации. Именно поэтому было принято решение о дополнительной настройке гиперпараметров этой модели. В целом эта настройка прогноз существенно не улучшила результат. «Случайный лес» всё-таки проигрывает алгоритму Линейной регрессии по качеству предсказания.

В итоге выведем сравнительный график (рис.19) результатов работы всех алгоритмов машинного обучения, где в качестве оцениваемого признака применялся только «Цена закрытия».

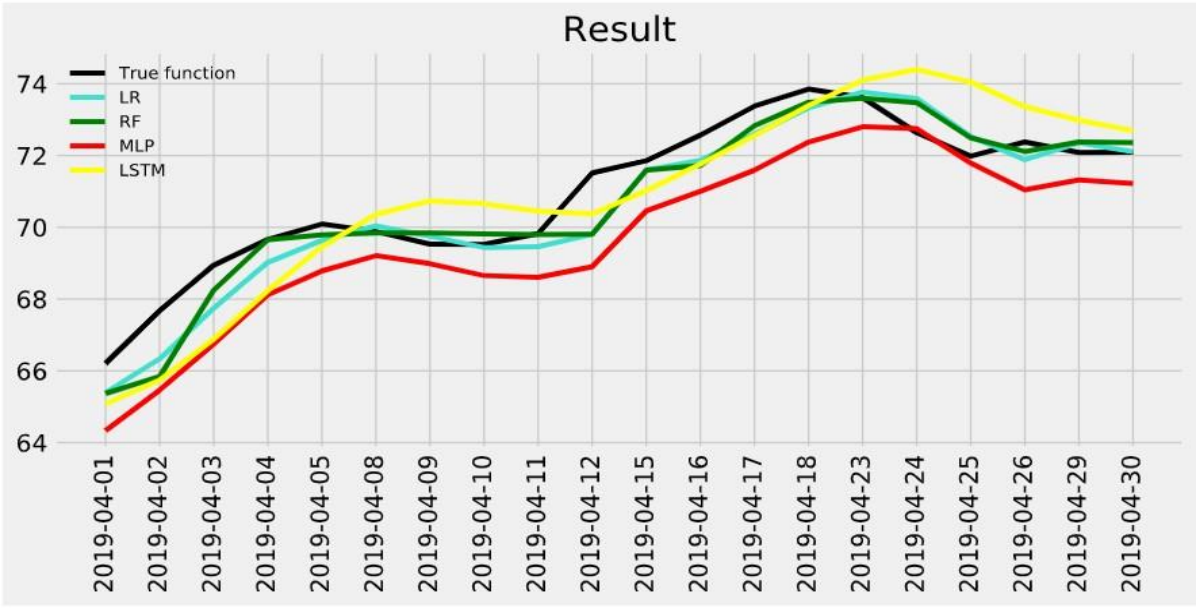

Рис.19. График работы всех алгоритмов с использованием одного признака

На этом рисунке видим, что выбранные архитектуры нейронных сетей, которые в большей степени подходили под данную задачу, предсказывают в целом хуже, чем классические алгоритмы МО.

Проверим качество работы Линейной регрессии с применением регуляризатора Ridge и без него, после добавления пяти дополнительных признаков OHLC и volume в модель, которые в теории должны влиять на конечную стоимость акций на фондовой бирже.

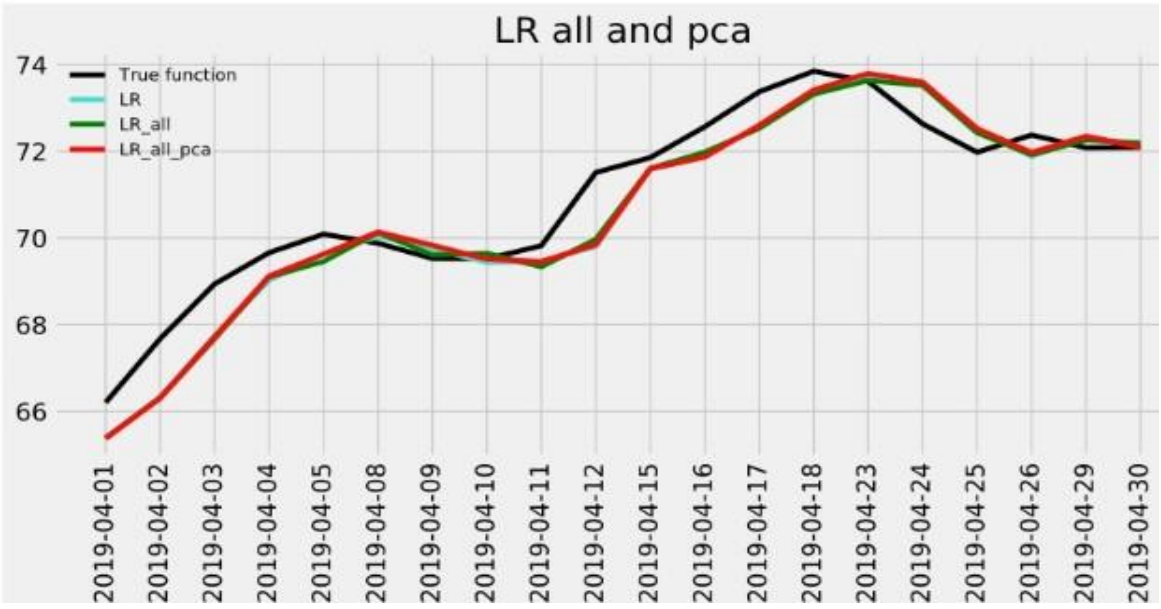

Рис.20. После применение PCA в модели LR

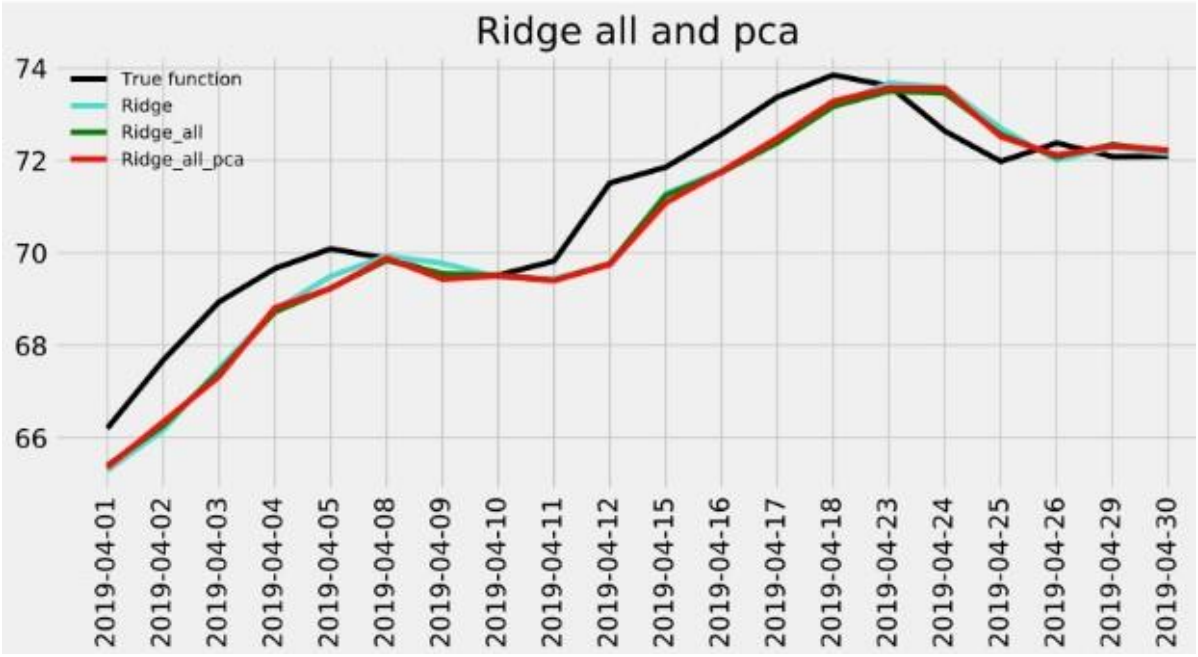

Рис.21. После применение PCA в модели Ridge

Стоит отметить, исходя из рис.20 и рис.21, что нормализация введённых признаков методом PCA в модель, не принесла желаемого результата. Модель линейной регрессии только ухудшила свой результат, и даже применение регуляризации его не улучшило, следовательно, можем сделать вывод о том, что регуляризацию для данных наборов данных использовать не стоит. Признаки, добавленные модель, будут только наносить вред качеству прогноза.

Влияние дополнительных признаков на прогноз средствами нейросетей

В качестве дополнительного исследования только уже возможностей нейронных сетей и качества их прогноза также добавим в нейросеть ранее введённые пять дополнительных признаков (high, low, open, close и volume). Выполним преобразования и построим график.

На рис.22 расшифруем график для MLP сети:

True function: Истинная кривая графика цен на акции
MLP: Кривая на графике только с признаком «цена закрытия»
MLP_all: Кривая, отражающая работу нейросети со всеми признаками

MLP_1: Кривая, отражающая работу нейросети без признака Close

MLP_2: Без признака - High

MLP_3: Без признака -Low

MLP_4: Без признака - Open

MLP_5: Без признака - Volume

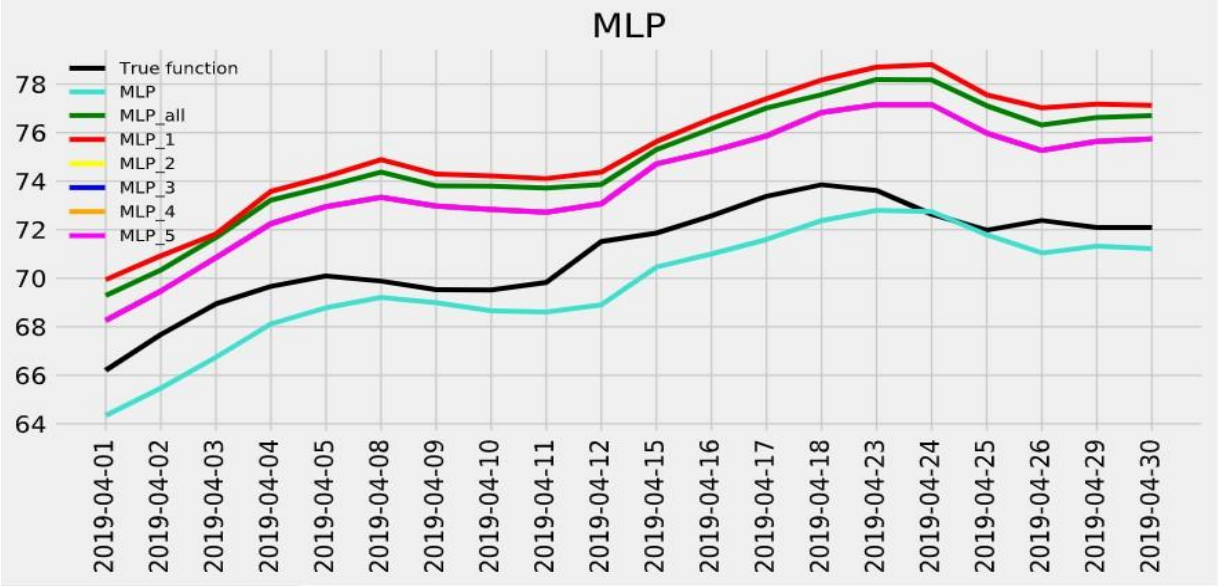

Рис.22.График работы MLP с признаками и без предобработки PCA

Далее на рис.23 расшифруем уже график LSTM-сети:

True function: Истинная кривая графика цен на акции

LSTM: Кривая на графике только с признаком «цена закрытия»

LSTM _all: Кривая, отражающая работу нейросети со всеми признаками

LSTM _2: Без признака - High

LSTM _3: Без признака - Low

LSTM _4: Без признака - Open

LSTM _5: Без признака - Volume

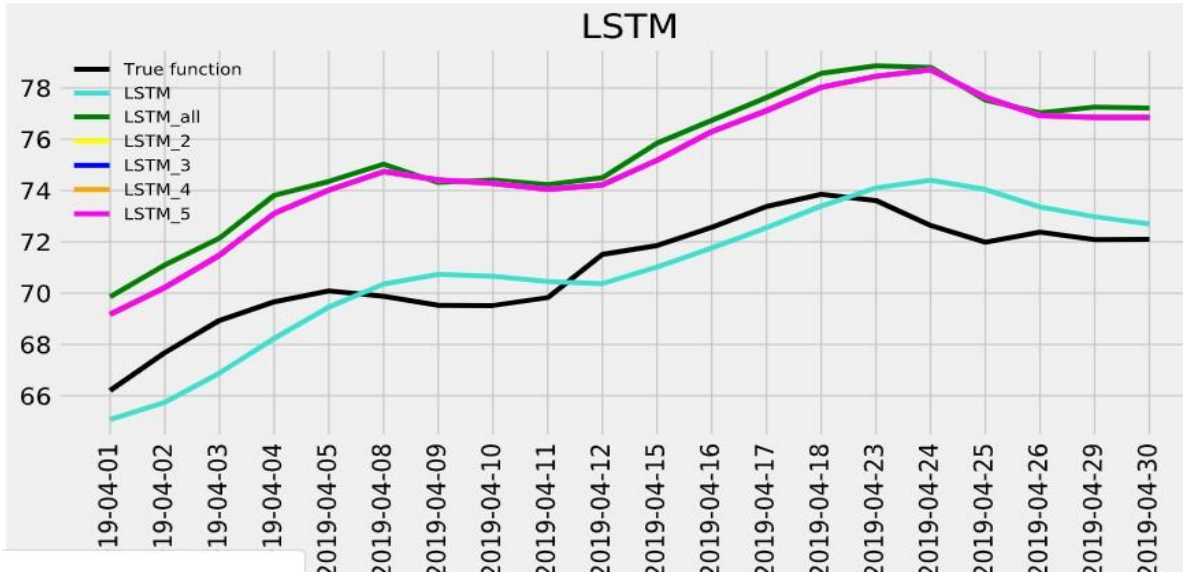

Рис.23.График работы LSTM с признаками и без предобработки PCA

Следует отметить, что на графике три кривые (LSTM_2, LSTM_3, LSTM_4) практически полностью совпадают.

Из графиков работ нейронных сетей вытекают предварительные выводы о том, что добавление дополнительных признаков, которые в теории должны были улучшить предсказания модели для технического анализа не эффективно, к примеру признаки OHLC: (high, low, open, close) линейно зависимы друг от друга. А индикатор volume хоть и не коррелируем с ними, но подвержен скачкам в разные дни, иногда практически без связи с самой ценой.

Чтобы избавиться от сильной корреляции признаков OHLC между собой, проведём дополнительную нормализацию данных. Для этого применим метод главных компонент (PCA). Он отлично подходит для коррелирующих друг с другом данных в признаках у модели и способен улучшить качество прогнозирования. С помощью метода PCA подвергнем преобразованию признаков OHLC в один.

На рис.24 расшифруем график отражающий работу MLP-сети:

True function: Истинная кривая графика цен на акции

MLP: Кривая на графике только с признаком «цена закрытия»

MLP_all: Кривая со всеми признаками без метода PCA

MLP _6: Учитываются только признак volume

MLP _pca: Учитываются только признаки OHLC с PCA

MLP_all_pca: Признаки OHLC с PCA и volume

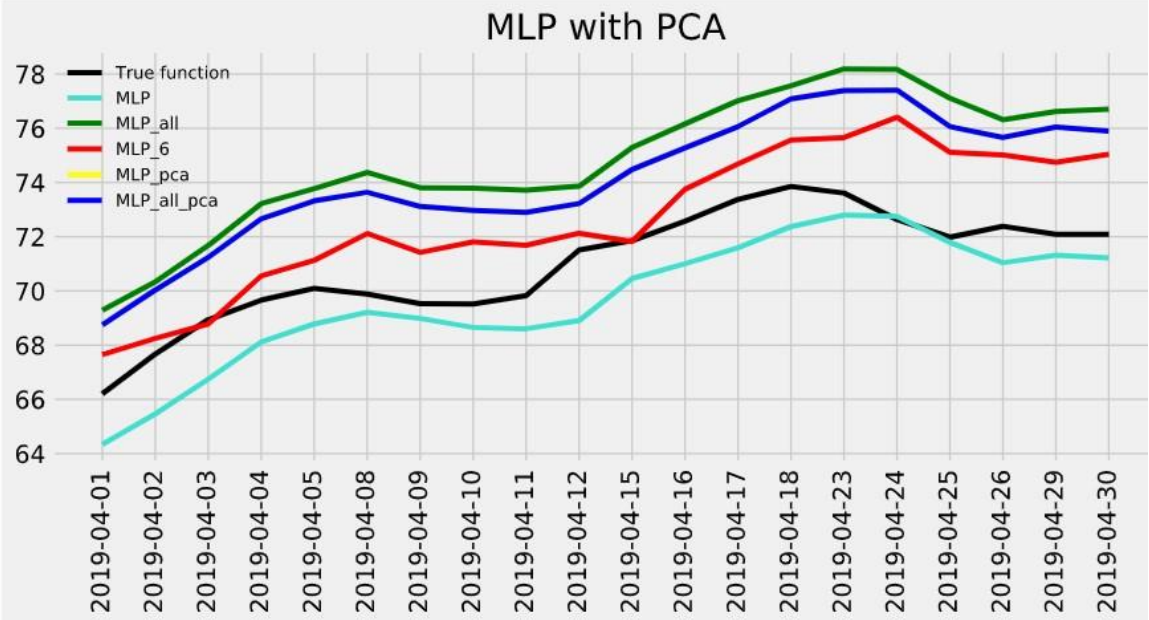

Рис.24.График работы MLP с признаками, с PCA

На рис.25 расшифруем теперь график отражающий работу LSTM-сети:

True function: Истинная кривая графика цен на акции

LSTM: Кривая на графике только с признаком «цена закрытия»

LSTM_all: Кривая со всеми признаками без PCA

LSTM _6: Учитываются только признак Volume

LSTM _pca: Учитываются только признаки OHLC с PCA

LSTM_all_pca: Признаки OHLC с PCA и Volume

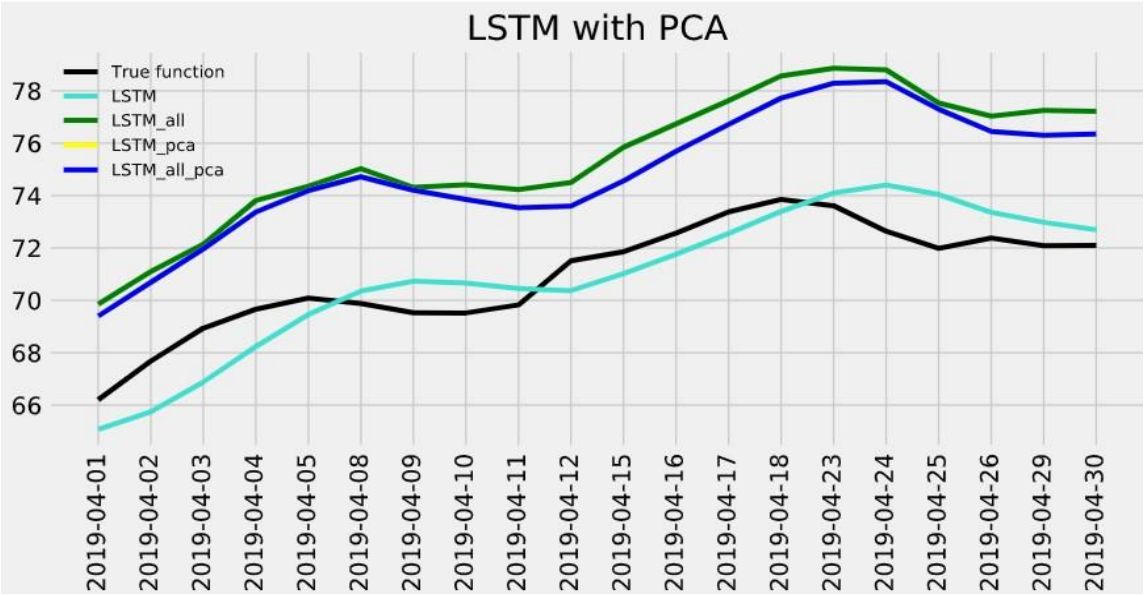

Рис.25.График работы LSTM с признаками, с PCA

Какие выводы из двух представленных выше графиков (рис.24, рис.25) можно сделать. После дополнительной нормализации признаков методом PCA имеется несущественное увеличение качества результата прогноза, но в целом добавление этих признаков в модель, которые в теории должны были улучшить прогноз, даже ухудшило его, даже после дополнительных преобразований с ними.

Выводы

В данной работе был проведён анализ различных методов машинного и, в частности, глубинного обучения.

Проведено исследование вводимых в работу дополнительных признаков в модель, на качество результатов прогнозов алгоритмов.

Произведён анализ библиотек и инструментов, для задачи прогнозирования движения цен на финансовых рынках. Осуществлён поиск, предобработка и нормализация исторических ценовых данных компании «BMW.DE» для обучения и тестирования модели.

Была разработана исследовательская программа для прогнозирования направления движения цен, реализующий архитектуру рекуррентной нейронной сети (RNN), построенной на элементах долгой краткосрочной памяти (LSTM) и многослойного перцептрона (MLP) и проведено сравнение результатов с классическими алгоритмами машинного обучения (Линейная регрессия, «Случайный лес», KNN).

Исходный код реализованного решения доступен по ссылке: https://github.com/Valentin26/Diplom2020.git

Заключение

Из полученных результатов следует, что точнее всего оказались модели LinearRegression и RandomForestRegressor. Причём модели Линейной регрессии именно без применения различных регуляризаций даже после тщательной нормализации данных. Возможно, такой результат был обусловлен спецификой исследуемых данных.

Выполненная работа показывает отсутствие паттернов в техническом анализе данных данного рода задач. Возможно такие паттерны имели место, если производилась бы классификация и к тому же брались, к примеру, минутные цены внутри одного дня и для них осуществлялись бы предсказания.

В результате исследования также можно отметить, что если в одном случае имеется картина прогноза за 10 дней и поведение цены произошло одним образом на 11-й день, то в следующий раз при повторении точно такой же ситуации она может повести себя совершенно случайным образом.

Поэтому, и нейронные сети, и обычные алгоритмы либо сильно ошибаются, либо наконец понимают, что самой выигрышной стратегией будет являться предсказание где цена останется на уровне предыдущего дня. То есть с равной вероятностью подъёма вниз или вверх. Тогда ошибка начнёт уменьшаться, и предсказания начнёт строиться по этому принципу.

Список литературы

1. Воронцов К. Курс лекций. Прогнозирование временных рядов //2019 URL: http://www.machinelearning.ru/wiki/images/3/3e/Voron-ML-Logic.pdf

2. TensorFlow (keras) - https://www.tensorflow.org/

3. Breiman L. Bagging predictors //Machine Learning.—1996.-Vol.24, no.2-Pp.123-140.

4. Michael Halls Moore //2016. (TSA) in Python - Linear Models to GARCH URL: http://www.blackarbs.com/blog/time-series-analysis-in-python-linear-models-to-garch/11/1/2016

5. RNN: URL https://en.wikipedia.org/wiki/Recurrent_neural_network

6. Бэстенс Д.-Э., Ван Ден Берг В.-М., Вуд Д. Нейронные сети и финансовые рынки: принятие решений в торговых операциях. Москва: ТВП, 1997.236с.

7. Джулли А., Пал С. Библиотека Keras - инструмент глубокого обучения //2017

8. Karpathy Andrej. The Unreasonable Effectiveness of Recurrent Neural Networks.May 21, 2015.
URL:http://karpathy.github.io/2015/05/21/rnneffectiveness/

9. Глубокое обучение, НЛП и репрезентации// 2014
URL: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

10. Brandon Amos. Image Completion with Deep Learning in TensorFlow //2016 URL: https://bamos.github.io/2016/08/09/deep-completion/

11. Jürgen Schmidhuber's page on. Recurrent Neural Networks //2017.
URL: http://people.idsia.ch/~juergen/rnn.html

12. Bas R. Steunebrink, Jürgen Schmidhuber. LSTM: A Search Space Odyssey.
URL: https://arxiv.org/abs/1503.04069

13. Michael Mathieu, Marc'Aurelio Ranzato. Learning Longer Memory in Recurrent Neural Networks.24 Dec 2014. URL: https://arxiv.org/abs/1412.7753

14. Петренко С. //2013. URL: datareview.info/article/issleduem-lstm-seti-chast-1/

15. Michael Nielsen. Neural Networks and Deep Learning.//2019 URL:http://neuralnetworksanddeeplearning.com/