Оглавление

[Глава 1. Построение теоретической базы 2](#__RefHeading___Toc2899_3736099160)

[1.1. Цель работы 2](#__RefHeading___Toc2901_3736099160)

[1.2. Введение 2](#__RefHeading___Toc2903_3736099160)

[1.3. Прогнозирование 3](#__RefHeading___Toc2905_3736099160)

[1.4. Временные ряды 3](#__RefHeading___Toc2907_3736099160)

[1.5. Оценка производительности (метрика) 5](#__RefHeading___Toc2909_3736099160)

[1.6. Рекуррентные нейронные сети 6](#__RefHeading___Toc2911_3736099160)

[1.6.1. Проблема долгосрочных связей 6](#__RefHeading___Toc2913_3736099160)

[1.6.2. Сети LSTM 7](#__RefHeading___Toc2915_3736099160)

[Глава 2. Описание моделей и методов прогнозирования 11](#__RefHeading___Toc2917_3736099160)

[2.1. Описание методов машинного обучения 11](#__RefHeading___Toc2919_3736099160)

[2.1.1. RandomForestRegressor 11](#__RefHeading___Toc2921_3736099160)

[2.1.2. CatBoost 12](#__RefHeading___Toc2923_3736099160)

[2.1.3. GradientBoostingRegressor 13](#__RefHeading___Toc2925_3736099160)

[2.1.4. SVR (Регрессия опорных векторов) 14](#__RefHeading___Toc2927_3736099160)

[2.1.5. LinearRegression 15](#__RefHeading___Toc2929_3736099160)

[2.1.6. KNeighborsRegressor 16](#__RefHeading___Toc2931_3736099160)

[Глава 3. Анализ и сравнение методов машинного обучения. 18](#__RefHeading___Toc2933_3736099160)

[3.1. Сравнение методов с помощью языка Python. 18](#__RefHeading___Toc2935_3736099160)

[3.1.1. Исходные данные 18](#__RefHeading___Toc2937_3736099160)

[3.1.2. Реализация методов на языке Python. 18](#__RefHeading___Toc2939_3736099160)

[3.2. Коэффициент детерминации - R2 20](#__RefHeading___Toc2941_3736099160)

[3.3. Средняя квадратическая ошибка - MSE 20](#__RefHeading___Toc2943_3736099160)

[3.4. Среднеквадратическая ошибка (RMSE) 21](#__RefHeading___Toc2945_3736099160)

[3.5. Среднее абсолютное отклонение - MAE 21](#__RefHeading___Toc2947_3736099160)

[3.6. Выбор метрики для анализа методов машинного обучения 21](#__RefHeading___Toc2949_3736099160)

[3.7. Сравнительный анализ методов машинного обучения. 22](#__RefHeading___Toc2951_3736099160)

[Вывод 23](#__RefHeading___Toc2953_3736099160)

[Приложение 23](#__RefHeading___Toc2955_3736099160)

# **Глава 1. Построение теоретической базы**

# **Цель работы**

Цель настоящей работы составляет изучение и освоение методов машинного обучения, анализ и сравнение эффективности существующих моделей прогнозирования и нейронной сети на открытых финансовых данных, а также рабочего веб-скрипта для их прогнозирования.

# **Введение**

Технология машинного обучения на основе анализа данных берёт начало в 1950 году, когда начали разрабатывать первые программы для игры в шашки. За прошедшие десятилетий общий принцип не изменился. Зато благодаря взрывному росту вычислительных мощностей компьютеров многократно усложнились закономерности и прогнозы, создаваемые ими, и расширился круг проблем и задач, решаемых с использованием машинного обучения.

Благодаря машинному обучению программист не обязан писать инструкции, учитывающие все возможные проблемы и содержащие все решения. Вместо этого в компьютер (или отдельную программу) закладывают алгоритм самостоятельного нахождения решений путём комплексного использования статистических данных, из которых выводятся закономерности и на основе которых делаются прогнозы.

Чтобы запустить процесс машинного обучение, для начала необходимо загрузить в компьютер Датасет (некоторое количество исходных данных), на которых алгоритм будет учиться обрабатывать запросы. Например, могут быть фотографии собак и котов, на которых уже есть метки, обозначающие к кому они относятся. После процесса обучения, программа уже сама сможет распознавать собак и котов на новых изображениях без содержания меток. Процесс обучения продолжается и после выданных прогнозов, чем больше данных мы проанализировали программой, тем более точно она распознает нужные изображения.

Благодаря машинному обучению компьютеры учатся распознавать на фотографиях и рисунках не только лица, но и пейзажи, предметы, текст и цифры. Что касается текста, то и здесь не обойтись без машинного обучения: функция проверки грамматики сейчас присутствует в любом текстовом редакторе и даже в телефонах. Причем учитывается не только написание слов, но и контекст, оттенки смысла и другие тонкие лингвистические аспекты. Более того, уже существует программное обеспечение, способное без участия человека писать новостные статьи (на тему экономики и, к примеру, спорта).

# **Прогнозирование**

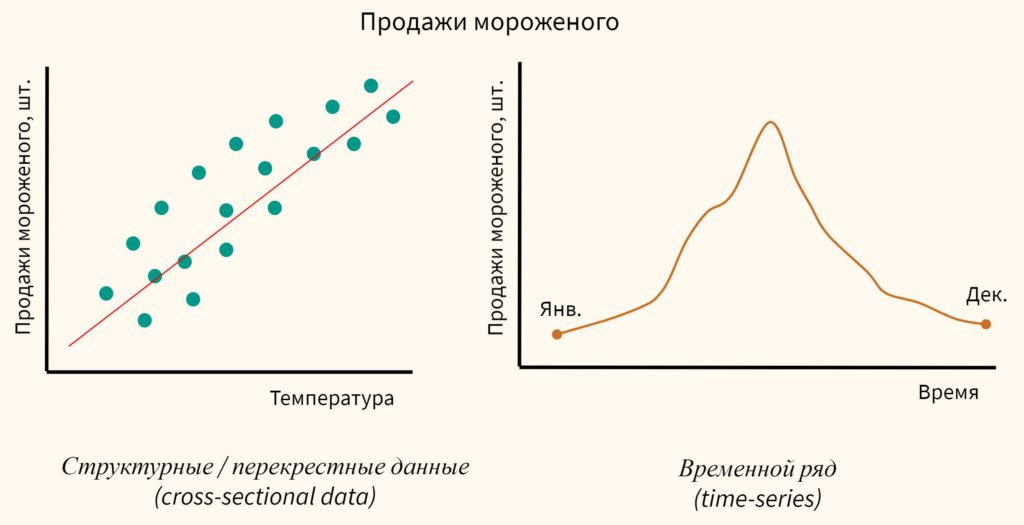
Прогнозирование — это задача науки о данных, которая является центральной для многих видов деятельности в организации. Например, крупные организации должны участвовать в планировании мощностей, чтобы эффективно распределять ограниченные ресурсы и ставить цели для измерения производительности относительно базового уровня.

Создание высококачественных прогнозов - непростая задача ни для машин, ни для большинства аналитиков. Полностью автоматические методы прогнозирования могут быть хрупкими и часто слишком негибкими, а аналитики, которые могут составлять высококачественные прогнозы, встречаются довольно редко, потому что прогнозирование — это специализированный навык науки о данных, требующий значительного опыта.

# **Временные ряды**

Временной ряд (time series) — это данные, последовательно собранные в регулярные промежутки времени. К таким данным относятся, например, цены на акции, объемы продаж чего-либо, изменения температуры с течением времени и т.д. Временные ряды очень часто строятся с помощью линейных диаграмм.

Посмотрим на изменение обычных данных и временных рядов.



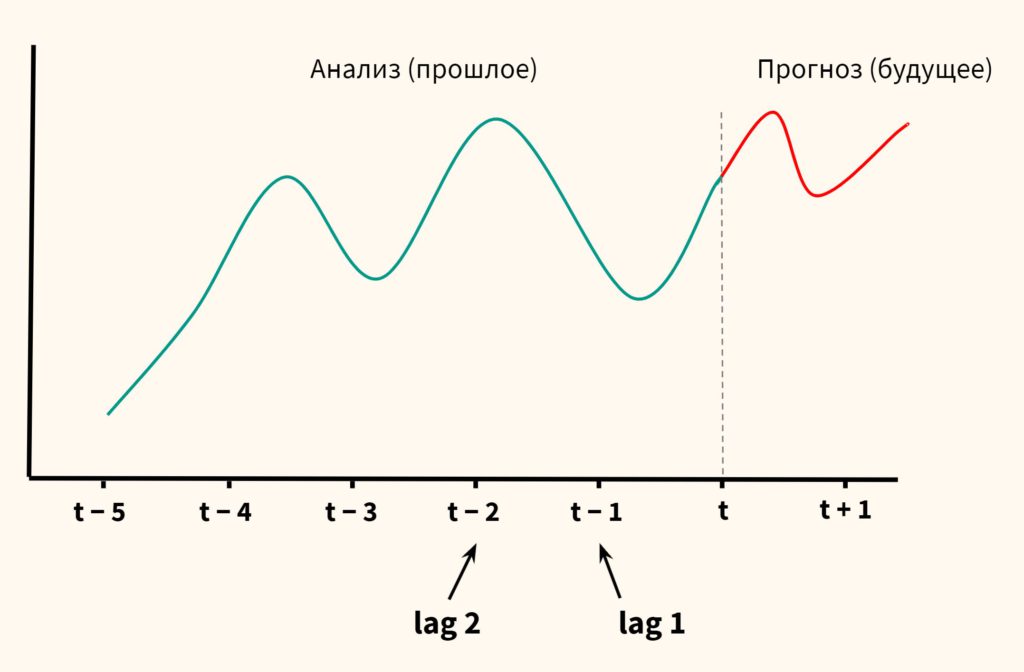
*Рис 1.4.1 Графики изменения обычных данных и временных рядов.*

Основное отличие графиков на рисунке 1: перекрестные данные предполагают независимость наблюдений, во временных рядах будущее зависит от прошлого.

Работа с временными рядами предполагает два аспекта:

1. Анализ временного ряда (time series analysis), т.е. понимание его структуры и закономерностей.
2. Моделирование и построение прогноза на будущее (time series forecasting).

Выполнив анализ временных рядов, мы должны спрогнозировать его, чтобы предсказать будущие значения, которые ряд будет принимать.



*Рис 1.4.2. График временного ряда.*

На рисунке 2 временем t обозначено настоящее время, t−1, t−2,… прошлое, t+1, t+2,… будущее. Понятие временного лага обозначено, как lag, означает запаздывания по сравнению с заданным периодом.

**Цель прогнозов:**

Поскольку прогнозирование временных рядов, таких как продажи и спрос, часто имеет невероятную коммерческую ценность, это увеличивает потребность в прогнозировании. Прогнозы обычно используются во многих производственных компаниях, поскольку оно управляет первичным бизнес-планированием, закупками и производственной деятельностью. Любые ошибки прогнозов будут проявляться по всей цепочке поставок или любой бизнес-структуре в этом отношении.

Таким образом, это важно для получения точных прогнозов, позволяющих сэкономить на затратах, и критически важно для успеха.

Прогнозирование временных рядов можно в общих чертах разделить на две категории:

1. Одномерное прогнозирование временных рядов. Это прогнозирование, в котором мы используем прежние значения временного ряда только для того, чтобы угадать предстоящие значения.
2. Прогнозирование многовариантных временных рядов. Это прогнозирование временных рядов, в котором мы используем для прогнозирования предикторы, отличные от ряда, также известные как экзогенные переменные.

# **Оценка производительности (метрика)**

Цель оценки производительности заключается в том, чтобы человек мог составить представление об эффективности модели и оценить ее. Метрика производительности показывает, насколько хорошо работает модель. Метрика позволяет оценить, насколько близки результаты модели к ожидаемым, причем исследователь сам определяет, насколько близкими должны быть эти значения. Выбор правильной метрики имеет решающее значение при оценке моделей машинного обучения.

Стоит также отметить, что метрика отличается от функции потерь, функции потерь — это функции, которые показывают меру производительности модели и используются для обучения модели машинного обучения (с использованием некоторой оптимизации), и обычно различаются по параметрам модели. С другой стороны, метрики используются для мониторинга и измерения производительности модели (во время обучения и тестирования) и не должны быть дифференцируемыми. Однако если для некоторых задач метрика производительности является дифференцируемой, ее можно использовать как в качестве функции потерь, так и метрики, такой как MSE.

Метрики эффективности — это способ показать, насколько точно модель отражает реальный мир. Они должны выбираться исходя из задачи, которую решает модель. В задаче и анализе можно применять несколько метрик эффективности, но также наряду с метриками эффективности есть и другие характеристики моделей - скорость обучения, скорость работы, надежность, робастность, интерпретируемость.

Метрики эффективности вычисляются как правило из двух векторов - предсказанных (теоретических) значений целевой переменной и эмпирических (реальных) значений. Обычно метрики устроены таким образом, что чем выше значение, тем модель лучше.

# **Рекуррентные нейронные сети**

Рекуррентные нейронные сети — вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки. В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому сети *RNN* применимы в таких задачах, где нечто целостное разбито на части, например: распознавание рукописного текста или распознавание речи. Было предложено много различных архитектурных решений для рекуррентных сетей от простых до сложных. В последнее время наибольшее распространение получили сеть с долговременной и кратковременной памятью (LSTM) и управляемый рекуррентный блок (GRU).

Существует много разновидностей, решений и конструктивных элементов рекуррентных нейронных сетей.

Трудность рекуррентной сети заключается в том, что если учитывать каждый шаг времени, то становится необходимым для каждого шага времени создавать свой слой нейронов, что вызывает серьёзные вычислительные сложности. Кроме того, многослойные реализации оказываются вычислительно неустойчивыми, так как в них как правило исчезают или зашкаливают веса. Если ограничить расчёт фиксированным временным окном, то полученные модели не будут отражать долгосрочных трендов. Различные подходы пытаются усовершенствовать модель исторической памяти и механизм запоминания и забывания.

# **Проблема долгосрочных связей**

Рекуррентная нейронная сеть используют полученную ранее информацию для решения последующих задач, например, уже полученные видеофрагменты для анализа последующих.

Иногда для решения задачи требуется просмотреть только последнюю информацию. Например, построить лингвистическую модель, которая пытается предсказать следующее слово, основываясь на предыдущих. Если мы пытаемся предсказать последнее слово в словосочетании «облака в небе», нам не нужен дополнительный контекст — очевидно, что следующее слово будет «небе». В тех случаях, когда разрыв между предыдущей информацией и местом, в котором она нужна, невелик, РНС справится с задачей.

**Но иногда требуется больше контекста.**Рассмотрим попытку предсказать последнее слово в тексте «Я вырос во Франции … Я свободно говорю на французском». Из предыдущих слов понятно, что следующим словом, вероятно, будет название языка, но, если мы хотим назвать правильный язык, нужно учесть упоминание о Франции. Разрыв между необходимой для учета информацией и точкой, в которой она нужна, становится большим.

К сожалению, **по мере увеличения этого разрыва РНС теряют связь между информацией.**

# **Сети LSTM**

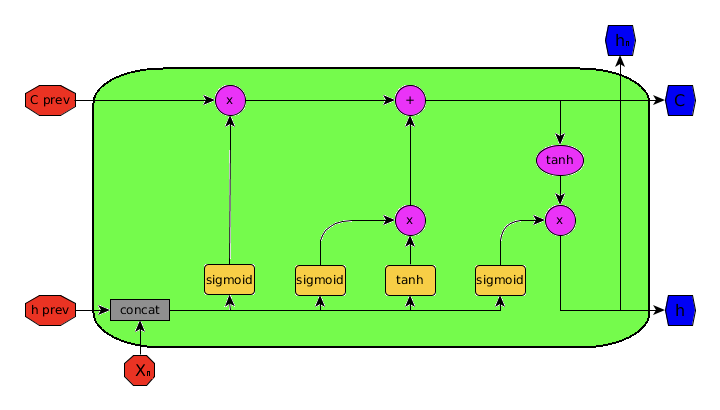
LSTM (long short-term memory, дословно (долгая краткосрочная память) — разновидность архитектуры [рекуррентных нейронных сетей](https://ru.wikipedia.org/wiki/Рекуррентная_нейронная_сеть), предложенная в [1997 году](https://ru.wikipedia.org/wiki/1997_год_в_науке) [Зеппом Хохрайтером](https://ru.wikipedia.org/w/index.php?title=Хохрайтер,_Зепп&action=edit&redlink=1) и [Юргеном Шмидхубером](https://ru.wikipedia.org/wiki/Шмидхубер,_Юрген).

Как и большинство рекуррентных нейронных сетей, LSTM-сеть является [универсальной](https://ru.wikipedia.org/wiki/Полнота_по_Тьюрингу) в том смысле, что при достаточном числе элементов сети она может выполнить любое вычисление, на которое способен обычный компьютер, для чего необходима соответствующая [матрица](https://ru.wikipedia.org/wiki/Матрица_(математика)) весов, которая может рассматриваться как программа.

В отличие от традиционных рекуррентных нейронных сетей, LSTM-сеть хорошо приспособлена к обучению на задачах [классификации](https://ru.wikipedia.org/wiki/Задача_классификации), обработки и [прогнозирования](https://ru.wikipedia.org/wiki/Задачи_прогнозирования) [временных рядов](https://ru.wikipedia.org/wiki/Временной_ряд) в случаях, когда важные события разделены временными лагами с неопределённой продолжительностью и границами. Относительная невосприимчивость к длительности временных разрывов даёт LSTM преимущество по отношению к альтернативным рекуррентным нейронным сетям, [скрытым марковским моделям](https://ru.wikipedia.org/wiki/Скрытая_марковская_модель) и другим методам обучения для последовательностей в различных сферах применения.

LSTM специально разработаны для устранения проблемы долгосрочной зависимости. Их специализация — запоминание информации в течение длительных периодов времени, поэтому их практически не нужно обучать.

LSTM сети были призваны решить проблему выбора того, какую информацию пропускать на следующий слой, а какую оставить. Далее схематично показана структура простейшей LSTM-сети.



*Рис. 1.1. Структура простейшей LSTM-сети.*

Здесь каждый входной элемент — вектор 𝑥, содержащий признаковое описание объекта последовательности.

h𝑝𝑟𝑒𝑣 — результат, получившийся на выходе предыдущей ячейки. h ∈ Rk, где 𝑘 — число нейронов скрытого слоя.

𝐶𝑝𝑟𝑒𝑣 — это дополнительный выход предыдущей ячейки. Он нужен, чтобы хранить информацию о предшествующих элементах последовательности. То есть его главное отличие от h в том, что он несет информацию, главным образом, не о предыдущей ячейки, но о большом числе, идущих до нее.

Конкатенацию здесь следует понимать, как соединение двух векторов в один, размерность которого, соответственно, равна сумме размерностей исходных.

Подробно разберем структуру сети.

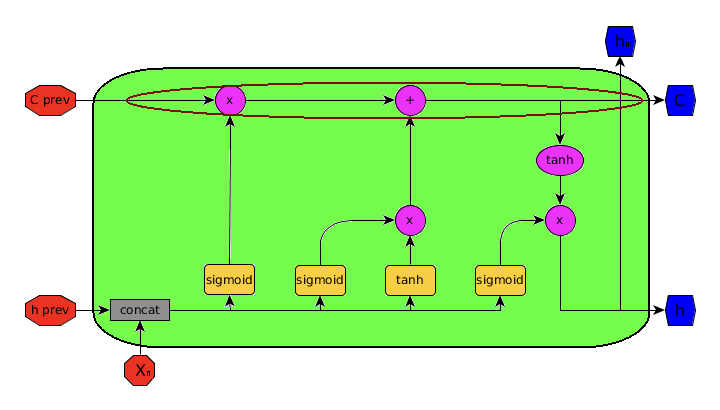
**Первый шаг**

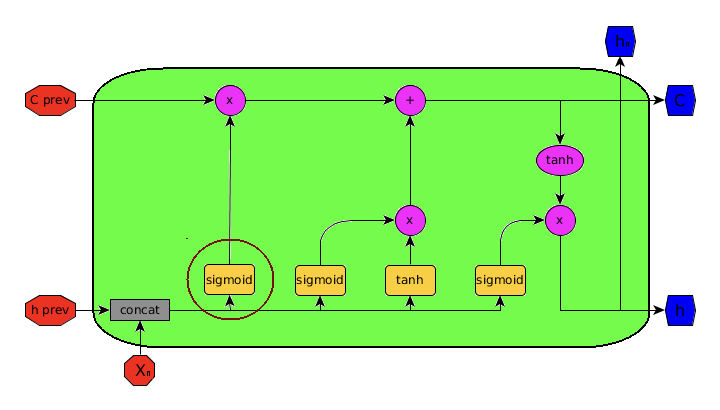
Одно из принципиальных отличий LSTM от остальных рекуррентных сетей — это последовательность слоев, которая на диаграмме изображена прямой линией в верхней части.

Через нее информация может спокойно проходить в следующую ячейку, при этом каждый новый элемент последовательности может воздействовать на нее по-разному, может поменять полностью, а может оставить без изменений.

Обратимся теперь к самим нейросетевым слоям. Рассмотрим первую сигмоиду.

Здесь сеть выбирает, какая информация о предшествующем элементе должна пройти через текущую ячейку. Для этого конкатенируются вектор выхода предыдущей ячейки h𝑡−1 и новый элемент последовательности 𝑥𝑡 и в качестве одного вектора пропускаются через слой с сигмоидной функцией активации. Назовем вектор, полученный после конкатенации 𝑣𝑡.

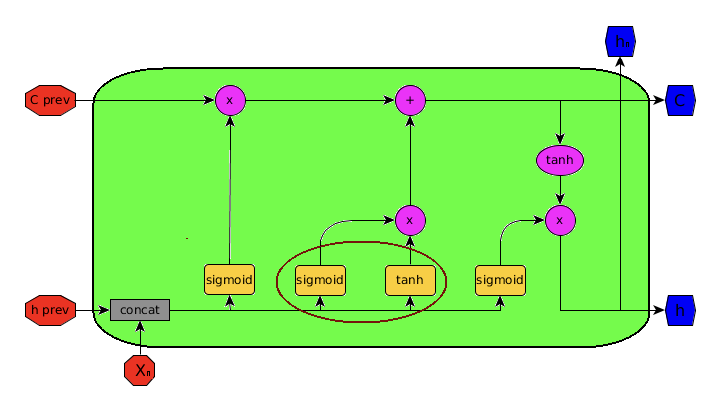




*Рис. 1.2. - 1.3. Первый этап работы LSTM сети*

**Второй шаг**

Теперь рассмотрим следующие два слоя. Они отвечают за то, какую новую информацию мы привнесем в модель.

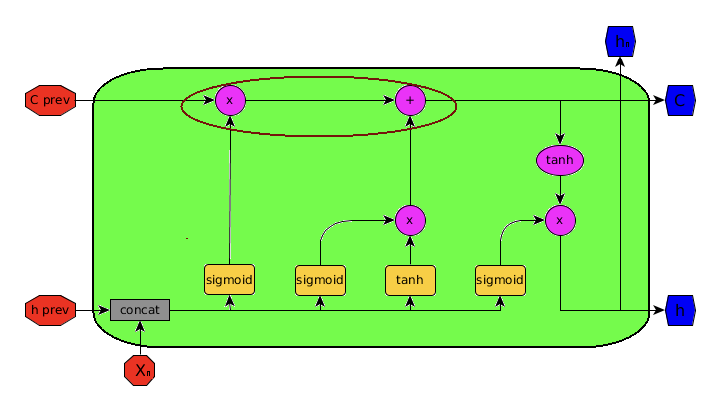


*Рис. 1.4. Второй этап работы LSTM сети*

Сигмоиду в этой структуре обычно называют “input gates layer”. Она решает, какие значения вектора будут обновлены. Затем слой с гиперболическим тангенсом определяет новых “кандидатов” для 𝐶. Далее, эти вектора поэлементно перемножаются, чтобы потом прибавиться к новому значению 𝐶.

**Третий шаг**

Теперь самое время обновить значения новые вектора 𝐶.



*Рис. 1.5. Третий этап работы LSTM сети*

На шаге 1 и 2 уже решено, какие значения будут записаны в 𝐶, остается лишь поэлементно домножить старое значение на то, что получено на шаге 2 и прибавить новые значения, полученные на шаге 2.

**Четвертый шаг**

Осталось на основе информации, которая хранится в 𝐶, а также 𝑣𝑡 получить выходное значение ячейки.

Для этого пропустим 𝑣𝑡 через слой с сигмоидной функцией активации, чтобы решить, что из 𝐶 стоит выдавать в качестве выходного значения. Потом применим слой с тангенциальной функцией активации к 𝐶 и умножим на выход сигмоидного слоя, чтобы получить только нужные значения. Полученный результат и будет выходным значением.

# **Глава 2. Описание моделей и методов прогнозирования**

# **Описание методов машинного обучения**

# **RandomForestRegressor**

**Случайный лес** — модель, состоящая из множества деревьев решений. Вместо того, чтобы просто усреднять прогнозы разных деревьев (такая концепция называется просто «лес»), эта модель использует две ключевые концепции, которые и делают этот лес случайным.

Случайная выборка образцов из набора данных при построении деревьев. При разделении узлов выбираются случайные наборы параметров.

В процессе тренировки каждое дерево случайного леса учится на случайном образце из набора данных. Выборка образцов происходит с возмещением (в статистике этот метод называется бутстреппинг, bootstrapping). Это даёт возможность повторно использовать образцы одним и тем же деревом. Хотя каждое дерево, может быть, высоко вариативным по отношению к определённому набору тренировочных данных, обучение деревьев на разных наборах образцов позволяет понизить общую вариативность леса, не жертвуя точностью.

При тестировании результат выводится путём усреднения прогнозов, полученных от каждого дерева. Подход, при котором каждый обучающийся элемент получает собственный набор обучающих данных (с помощью бутстреппинга), после чего результат усредняется, называется бэггинг (bagging, от bootstrap aggregating).

Вторая базовая концепция случайного леса заключается в использовании определённой выборки параметров образца для разделения каждого узла в каждом отдельном дереве. Обычно размер выборки равен квадратному корню из общего числа параметров. То есть, если каждый образец набора данных содержит 16 параметров, то в каждом отдельном узле будет использовано 4. Хотя обучение случайного леса можно провести и с полным набором параметров, как это обычно делается при регрессии. Этот параметр можно настроить в реализации случайного леса в Scikit-Learn.

Случайный лес сочетает сотни или тысячи деревьев принятия решений, обучая каждое на отдельной выборке данных, разделяя узлы в каждом дереве с использованием ограниченного набора параметров. Итоговый прогноз делается путём усреднения прогнозов от всех деревьев.

Алгоритм построения случайного леса, состоящего из N деревьев, выглядит следующим образом:

Для каждого n=1,…,N:

Сгенерировать выборку Xn c помощью бутстрэпа;

Построить решающее дерево bn по выборке Xn:

— по заданному критерию мы выбираем лучший признак, делаем разбиение в дереве по нему и так до исчерпания выборки

— дерево строится, пока в каждом листе не более nmin объектов или пока не достигнем определенной высоты дерева

— при каждом разбиении сначала выбирается m случайных признаков из n исходных, и оптимальное разделение выборки ищется только среди них.

Итоговый классификатор

Где N – количество деревьев;

i – счетчик для деревьев;

b – решающее дерево;

x – сгенерированная нами на основе данных выборка.

Простыми словами — для задачи классификации мы выбираем решение голосованием по большинству, а в задаче регрессии — средним.

Рекомендуется в задачах классификации брать , а в задачах регрессии , где n — число признаков. Также рекомендуется в задачах классификации строить каждое дерево до тех пор, пока в каждом листе не окажется по одному объекту, а в задачах регрессии — пока в каждом листе не окажется по пять объектов.

Таким образом, случайный лес — это бэггинг над решающими деревьями, при обучении которых для каждого разбиения признаки выбираются из некоторого случайного подмножества признаков.

# **CatBoost**

**CatBoost** — это платформа GBDT, основанная на симметричных деревьях решений (деревья забвения), которая реализует меньшее количество параметров, поддерживает категориальные переменные и обеспечивает высокую точность. Основной проблемой является эффективная и разумная обработка категориальных функций. Как видно из названия, CatBoost состоит из категорий и повышения. Кроме того, CatBoost также решает проблемы смещения градиента (Gradient Bias) и смещения предсказания (Сдвиг предсказания), тем самым уменьшая вероятность переобучения, тем самым повышая точность и способность алгоритма к обобщению.

CatBoost — библиотека для градиентного «бустинга», главным преимуществом которой является то, что она одинаково хорошо работает «из коробки» как с числовыми признаками, так и с категориальными.

Программное обеспечение разработано по методологии SCRUM.

Алгоритм работы, следующий: для каждого документа имеется набор значений признаков, имеется дерево, в вершинах дерева — условия. Если условие выполнено, осуществляется переход в правую ветвь вершины, иначе в левого. Нужно пройти до листа по дереву в соответствии со значениями признаков для документа. На выходе каждому документу соответствует значение листа. Это и есть ответ.

Идея «бустинг» - подхода заключается в комбинации слабых (с невысокой обобщающей способностью) функций, которые строятся в ходе итеративного процесса, где на каждом шаге новая модель обучается с использованием данных об ошибках предыдущих. Результирующая функция представляет собой линейную комбинацию базовых, слабых моделей.

Итого при достаточно большом количестве деревьев можно сильно уменьшить ошибку, однако не стоит забывать, что чем больше деревьев, тем дольше обучается модель и в какой-то момент прирост качества становится незначительным.

В основе CatBoost лежит градиентный «бустинг».

Градиент функции ошибки — все производные по всем значениям функции

Градиентный «бустинг» — метод машинного обучения, который создает решающую модель прогнозирования в виде ансамбля слабых моделей прогнозирования, обычно деревьев решений. Он строит модель поэтапно, позволяя оптимизировать произвольную дифференцируемую функцию потерь.

# **GradientBoostingRegressor**

**GradientBoosting** — это метод машинного обучения для задач регрессии, классификации и других задач, которые создают модель прогнозирования, обычно деревьев решений. Когда дерево решений сложно обучаемо, используется GradientBoosting, который обычно эффективней, чем randomforest. Он строит модель поэтапно, как и другие bosting-методы, и обобщает их, позволяя оптимизировать произвольные дифференцируемые функция потерь.

Как и другие boosting-методы, GradientBosting объединяет «сложно обучаемые» деревья решений в одно «легко обучаемое» дерево решений итеративным способом. Это проще всего объяснить с помощью метода регрессии наименьших квадратов, цель которого - «научить» модель прогнозировать значение формы минимизируя среднеквадратичную ошибку

где индексы некоторой базы данных для обучения размерности n фактических значений выходной переменной y:

- прогнозируемое значение

y - наблюдаемое значение

n - количество образцов в y

Принцип работы GradientBoostingRegressor можно рассмотреть на примере с M количеством этапов. Если обучить единственное решающее дерево, то качество такой модели, скорее всего, будет низким. Однако о построенном дереве будет известно, на каких объектах оно давало точные предсказания, а на каких ошибалась. Необходимо использовать эту информацию и обучить еще одну модель. На каждом этапе GradientBoosting предположим некоторую несовершенную модель (для низкого значения m эта модель может просто вернуть , где RHS - среднее значение y). Чтобы улучшить наш алгоритм должен добавить новую оценку . Таким образом,

или

Следовательно, GradientBoosting получит остаток h, который поможет уменьшить ошибку композиции:

Как и в других вариантах Boosting - методов каждый пытается исправить ошибки . Обобщение этой идеи на функции потерь, отличные от квадратичной ошибки, а также на задачи классификации и ранжирования следует из наблюдения, что остатки для данной модели отрицательные градиенты функции потерь среднеквадратичной ошибки (MSE).

# **SVR (Регрессия опорных векторов)**

**Регрессия опорных векторов** — это модель машинного обучения, в которой для прогнозирования непрерывной переменной используется машина опорных векторов, алгоритм классификации. В то время как модели линейной регрессии минимизируют ошибку между фактическими и прогнозируемыми значениями с помощью линии наилучшего соответствия, SVR удается подобрать лучшую линию в пределах пороговых значений, иначе приемлемый предел погрешности. Это трубка, содержащая допустимую погрешность нашей модели, где любые ошибки внутри трубки наименее важны. Однако учитываются точки за пределами трубы, а расстояние между точкой и самой трубкой измеряется и обозначается как опорный вектор.

SVR позволяет нам выбирать, насколько наша модель терпима к ошибкам, через приемлемый предел погрешности и через допуск выхода за пределы погрешности. Формулировку задачи SVR часто лучше всего выводить с геометрической точки зрения. Аппроксимируемую функцию с непрерывными значениями можно записать в виде уравнении:

Для многомерных данных мы увеличиваем x на единицу и включаем b в вектор w в простую математическую запись и получаем многомерную регрессию в уравнении:

SVR формулирует эту задачу аппроксимации функции как задачу оптимизации, которая пытается найти самую узкую трубку с центром вокруг поверхности, минимизируя ошибку прогнозирования, то есть расстояние между прогнозируемым и желаемым выходными данными. Первое условие дает целевую функцию в уравнении:

где || w || - величина вектора нормали к аппроксимируемой поверхности.

# **LinearRegression**

Модель зависимости переменной x от одной или нескольких других переменных (факторов, регрессоров, независимых переменных) с линейной функцией зависимости.

Математическое представление линейной регрессии достаточно легко для восприятия. Линия простой линейной регрессии задается уравнением вида

,

где x – независимая переменная, а y – зависимая.

Цель алгоритма линейной регрессии – установить такие коэффициенты, чтобы стало возможно определить данную регрессионную модель, а достигается это в процессе обучения. Для этого существует целый ряд методов, однако наиболее популярные из них — это метод обыкновенных наименьших квадратов и градиентный спуск.

Метод обыкновенных наименьших квадратов (Ordinary Least Squares) предназначен для проведения линии через разброс точек так, чтобы сумма квадратов отклонений точек от линии была минимальной. Простыми словами: ошибка (остаточное значение, отклонение) в данном методе — не что иное, как разница между «реальностью и ожиданием», полученным значением y и прогнозируемым значением ŷ.

(Реальное значение — Прогнозированное значение).

Линейная регрессия относится к задаче определения «линии наилучшего соответствия» через набор точек данных и стала простым предшественником нелинейных методов, которые используют для обучения нейронных сетей.

# **KNeighborsRegressor**

**kNN** – метрический̆ метод классификации, то есть объекты представляются в виде точек в пространстве и между ними вычисляются расстояния. Следовательно, признаки должны быть нормированы (приведены к одному масштабу) Число k – входной̆ параметр алгоритма и может быть оптимально настроен.

Он используется для классификации и регрессии. В обоих случаях входные данные состоят из k ближайших обучающих примеров в наборе данных. Результат зависит от того, используется ли k -NN для классификации или регрессии.

В классификации k-NN выходом является принадлежность к классу. Объект классифицируется множеством «голосов» его «соседей», причем объект назначается классу, наиболее распространенному среди его ближайших k «соседей» (k - положительное целое число, обычно небольшое). Если k = 1, то объект просто присваивается классу этого единственного ближайшего «соседа».

В регрессии k-NN выходом является значение свойства объекта. Это значение является средним из значений k ближайших «соседей». k -NN — это тип классификации, при котором функция аппроксимируется только локально, а все вычисления откладываются до оценки функции. Поскольку этот алгоритм основан на расстоянии для классификации, если признаки представляют разные физические единицы или имеют совершенно разные масштабы, то нормализация обучающих данных может значительно повысить его точность

Примеры обучения — это векторы в многомерном пространстве признаков, каждый из которых имеет метку класса. Фаза обучения алгоритма состоит только из сохранения векторов признаков и меток классов обучающих выборок.

Обычно используемой метрикой расстояния для непрерывных переменных является евклидово расстояние. Для дискретных переменных, например для классификации текста, можно использовать другую метрику, например метрику перекрытия (или расстояние Хэмминга). В контексте данных микроматрицы экспрессии генов, например, k -NN использовался с коэффициентами корреляции, такими как Пирсон и Спирмен, в качестве показателя.

Евклидово расстояние является наиболее распространенной метрикой расстояния, измеряющей абсолютное расстояние между точками в многомерном пространстве. Формула выглядит следующим образом:

В обычном случае используют так называемое простое невзвешенное голосование (simple unweighted voting). При это предполагается, что все k примеров имеют одинаковое право «голоса» независимо от расстояния до классифицируемого объекта.

Однако, логично предположить, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса. Поэтому для улучшения результатов классификации вводят взвешивание примеров в зависимости от их удалённости. В этом случае используют взвешенное голосование (weighted voting). В основе идеи взвешенного голосования лежит введение «штрафа» для класса, в зависимости от того, насколько относящиеся к нему примеры удалены от классифицируемого объекта. Такой «штраф» представляется как сумма величин, обратных квадрату расстояний от примера jj-го класса до классифицируемого объекта (часто данное значение называют показателем близости):

​

где D — оператор вычисления расстояния, x — вектор признаков классифицируемого объекта, aij ​ — i-й пример j-го класса.

Таким образом, «побеждает» тот класс, для которого величина Qj ​окажется наибольшей. При этом также снижается вероятность того, что классы получат одинаковое число голосов.  
 k - NN — это частный случай "баллонной" оценки плотности ядра с переменной полосой пропускания и однородным ядром.

Наивную версию алгоритма легко реализовать, вычислив расстояния от тестового примера до всех сохраненных примеров, но это требует больших вычислительных ресурсов для больших обучающих наборов. Использование приближенного алгоритма поиска ближайшего соседа делает k- NN вычислительно управляемым даже для больших наборов данных. Многие алгоритмы поиска ближайшего соседа были предложены на протяжении многих лет; они, как правило, стремятся сократить количество фактически выполняемых оценок расстояния.

k - NN имеет некоторые сильные результаты согласованности. Поскольку количество данных приближается к бесконечности, двухклассовый алгоритм k- NN гарантированно дает частоту ошибок не хуже, чем удвоенную частоту ошибок Байеса (минимально достижимую частоту ошибок с учетом распределения данных).

Различные улучшения скорости k -NN возможны с помощью графов близости.

# **Глава 3. Анализ и сравнение методов машинного обучения.**

# **Сравнение методов с помощью языка Python.**

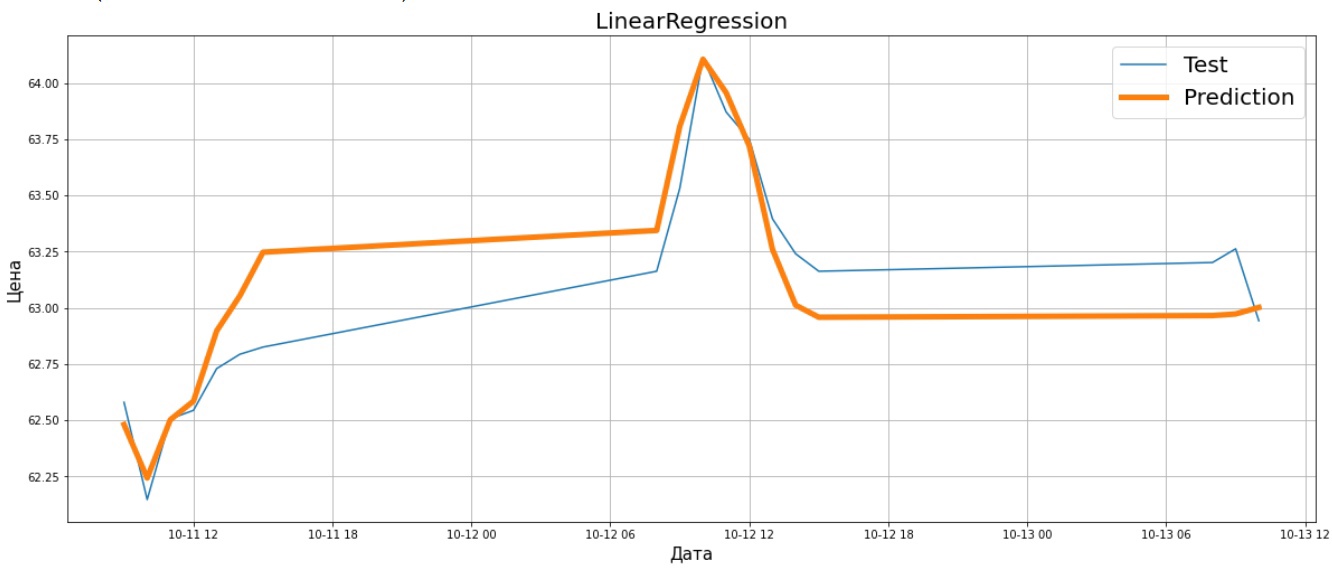
# **Исходные данные**

Для сравнительного анализа, рассмотрим работу наших методов на данных об изменении цен валюты EURO.

# **Реализация методов на языке Python.**

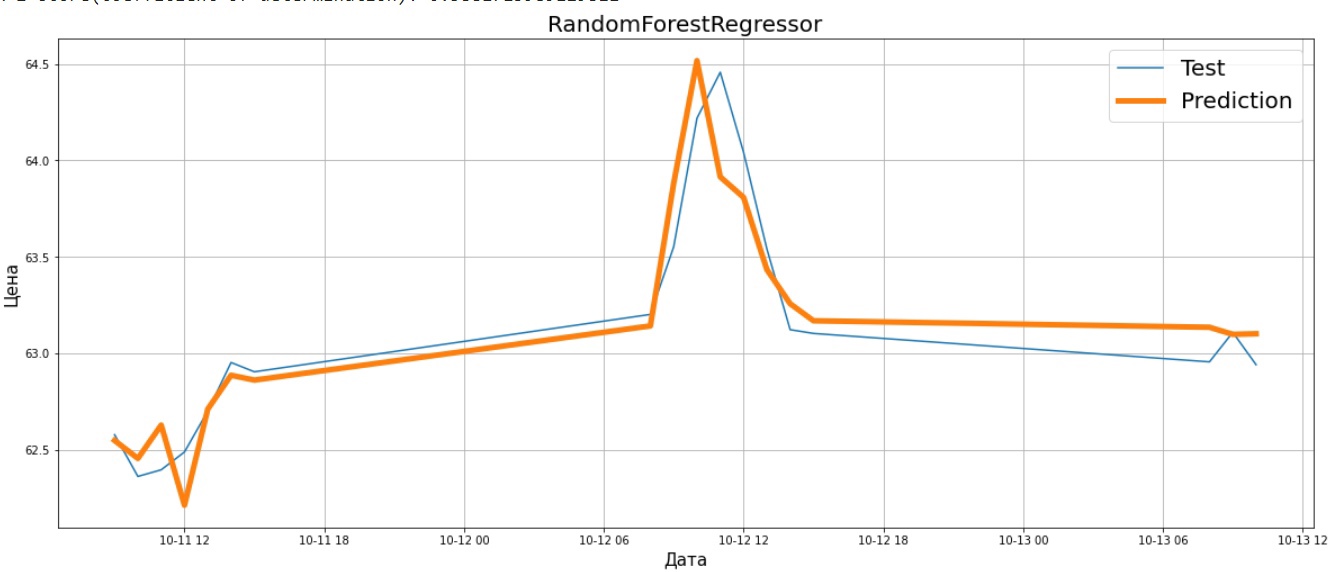
На вход модели подадим данные об изменении цен валюты EURO за период с 01.10.2022 по 13.10.2022 год в соотношении 80 % модель получает на обучение, 20% прогнозирует сама.

Результаты работы методов машинного обучения выведены на графики 3.1 - 3.5. Prediction(цена) (линия оранжевого цвета) на графике показывает результаты работы обучающей модели, линия цены (голубой цвет) выводит данные, полученные моделью на вход.



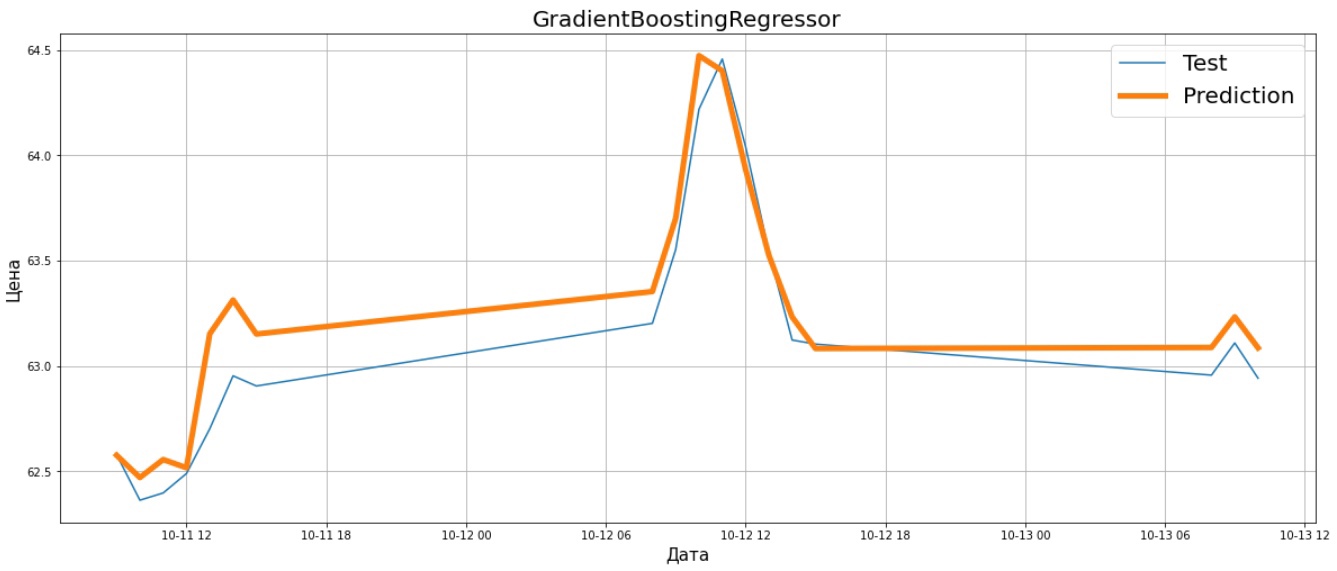
*R2 = 0,821*

График 3.1. Прогнозирование цены валюты EURO методом линейной регрессии.



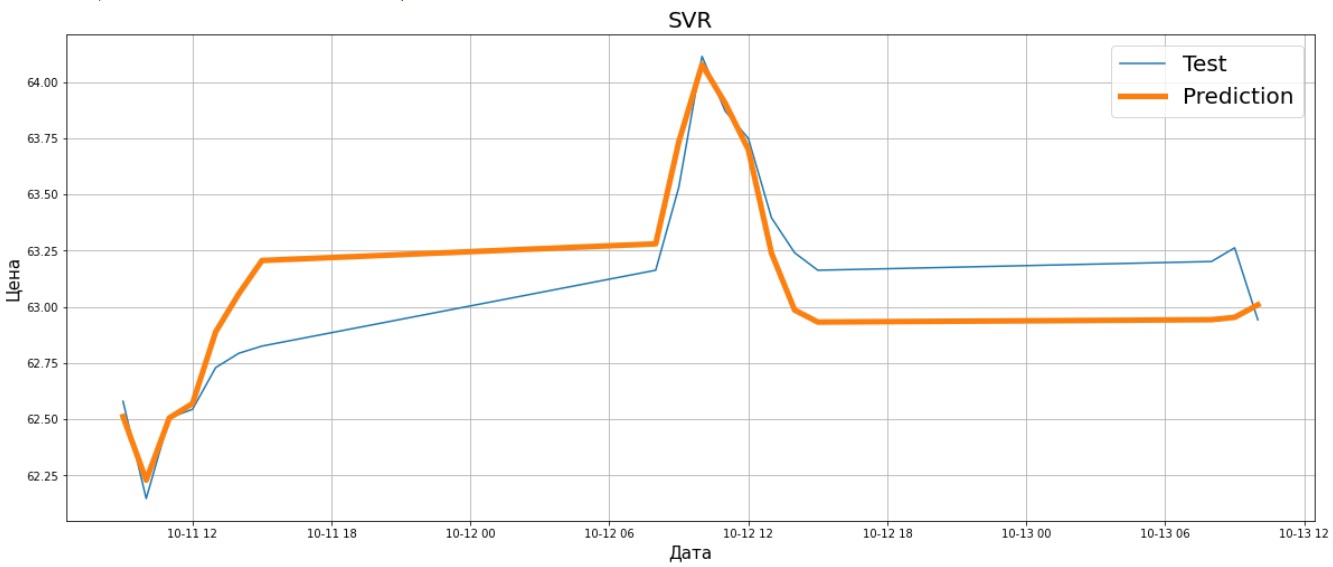
*R2 = 0,914*

График 3.2. Прогнозирование цены валюты EURO методом RandomForestRegressor.



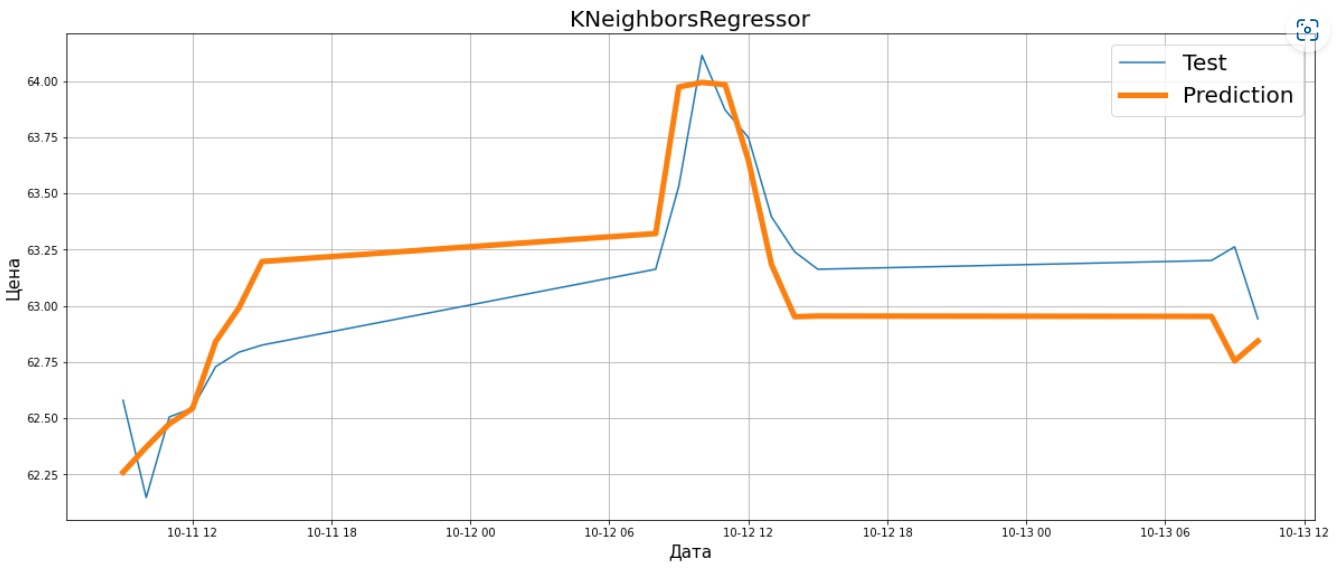
*R2 = 0,895*

График 3.3. Прогнозирование цены валюты EURO методом GradientBoostingRegressor.



*R2 = 0,853*

График 3.4. Прогнозирование цены валюты EURO методом SVR.



*R2 = 0,834*

График 3.5. Прогнозирование цены валюты EURO методом KneighborsRegressor.

При анализе полученных графиков, мы можем сделать вывод, что из-за прогнозирования большого промежутка времени коэффициент детерминации низкий, а часто даже отрицательный. Для того, чтобы прогноз был более точный мы значительно снизили количество прогнозируемых дней и провели анализ оптимального количества прогнозируемых дней для каждого из методов.

# **Коэффициент детерминации - R2**

R2 – это коэффициент линейной детерминации. Показывает, во сколько предсказанные нашей моделью данные лучше, чем среднее значение. Чем ближе к 1, тем лучше работает модель.

Где *MSE (model)* – среднеквадратичное отклонение между фактическими и предсказанными данными

*MSE (baseline)* - среднеквадратичное отклонение между фактическими данными и средним значением

Характеризует степень сходства исходных данных и предсказанных. В отличии от средней квадратической ошибки не зависит от единиц измерения данных, поэтому поддается сравнению

Когда R² отрицательно, это означает, что модель хуже, чем предсказание среднего значения

Эта метрика не определена, если y = const. Надо следить, чтобы в выборке присутствовали разные значения целевой переменной.

# **Средняя квадратическая ошибка - MSE**

MSE в основном измеряет среднеквадратичную ошибку наших прогнозов. Для каждой точки вычисляется квадратная разница между прогнозами и целью, а затем усредняются эти значения.

* ŷ i - прогнозируемое значение для i -го наблюдения
* y i - наблюдаемое значение для i -го наблюдения
* n - размер выборки

Чем выше это значение, тем хуже модель. Он никогда не бывает отрицательным, поскольку мы возводим в квадрат отдельные ошибки прогнозирования, прежде чем их суммировать, но для идеальной модели это будет ноль.

# **Среднеквадратическая ошибка (RMSE)**

RMSE — это просто квадратный корень из MSE. Квадратный корень введен, чтобы масштаб ошибок был таким же, как масштаб целей.

Метрика, которая сообщает нам квадратный корень из средней квадратичной разницы между прогнозируемыми значениями и фактическими значениями в наборе данных. Чем ниже RMSE, тем лучше модель соответствует набору данных.

# **Среднее абсолютное отклонение - MAE**

Средняя абсолютная ошибка (или среднее абсолютное отклонение) — это другая метрика, которая определяет среднее абсолютное расстояние между прогнозируемыми и целевыми значениями. MAE определяется следующим образом:

* ŷ i - прогнозируемое значение для i -го наблюдения
* y i - наблюдаемое значение для i -го наблюдения
* n - размер выборки

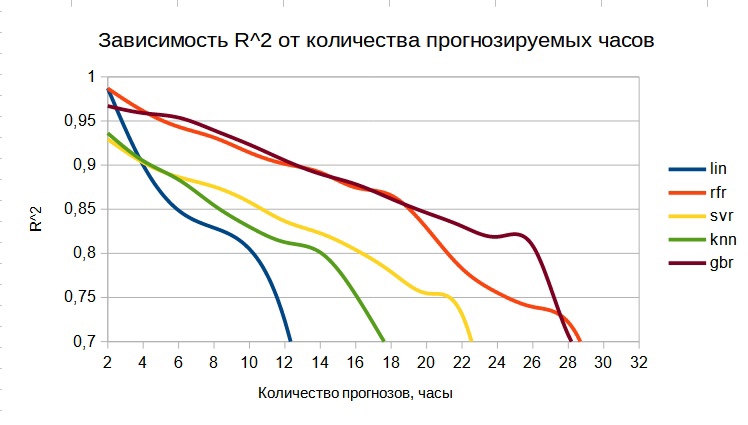
MAE более устойчив к выбросам, чем MSE. Основная причина в том, что в MSE путем возведения в квадрат ошибок, выбросы получают больше внимания и доминируют в окончательной ошибке и влияют на параметры модели.

# **Выбор метрики для анализа методов машинного обучения**

𝑅2 значение очень интуитивно понятно. Но исследования показывают, что 𝑅2 действителен только для линейной регрессии. Однако большинство моделей регрессии, такие, например, как дерево решений или KNN, являются нелинейными моделями. Для нелинейных моделей мы не можем полностью доверять 𝑅2.

Предпочтительно всегда использовать 𝑅2 вместе с другими показателями, такими как MAE и RMSE. Когда необходимо уменьшить влияние выбросов, лучше использовать MAE, когда выбросы нельзя игнорировать, лучше использовать RMSE.

# **Сравнительный анализ методов машинного обучения.**



*График 3.4.1. Зависимость коэффициента детерминации от количества прогнозируемых часов*

Данный проект посвящен практической реализации моделей линейной регрессии, RandomForestRegressor, GradientBoostingRegressor, SVR, KneighborsRegressor, CatBoostRegressor на языке программирования Python и сравнению эффективности данных моделей.

Для того, чтобы выяснить оптимальное количество прогнозируемых дней для каждой из моделей было проанализировано изменение коэффициента детерминации.

В ходе проекта была проанализирована работа моделей на открытых финансовых данных валюты EURO за период с 01.10.2022 по 13.10.2022 год. Для более точного изучения и сравнения моделей подавались исходные данные в разных процентных соотношениях «обучение – прогноз». На графиках 3.1 – 3.5 можно увидеть результаты работы моделей в соотношении данных 80% - обучение и 20% - прогнозирование.

Полученные графики показывают, что из рассмотренных методов, RandomForestRegressor и GradientBoostingRegressor наиболее точно прогнозируют изменение цены валюты EURO в соответствии с реальными данными.

Исходя из результатов, показанных на графике 3.4.1., можно сделать вывод, что методы Random Forest и Gradient Boosting очень близки, по своим прогнозам, но все-таки Gradient Boosting показал чуть более точные результаты.

# **Вывод**

В ходе выполнения проекта были изучены и освоены методы машинного обучения, а также произведено сравнение эффективности существующих моделей прогнозирования на открытых финансовых данных криптовалюты Bitcoin и компании Microsoft. Была реализована нейронная сеть LSTM, которая так же была протестирована в целях прогнозирования.

# **Приложение**

Приложение(код) часть 1

!pip install yfinance

!pip install statsmodels

!pip install arch

!pip install keras

import pandas as pd

import yfinance as yf

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

from tqdm import tqdm\_notebook as tqdm

from data\_preparation import \*

from scipy.stats import pearsonr

from sklearn.metrics import r2\_score

import itertools

from sklearn.metrics import mean\_squared\_error as mse

def get\_stocks(ticker, start = &quot;1900-01-01&quot;, end = None):

tickerData = yf.Ticker(tickerSymbol)

df = tickerData.history(period=&#39;1d&#39;, start=start, end=end)

if &quot;Stock Splits&quot; in df.columns:

df.drop([&quot;Stock Splits&quot;], axis = 1, inplace = True)

if &quot;Dividends&quot; in df.columns:

df.drop([&quot;Dividends&quot;], axis = 1, inplace = True)

#if &quot;Volume&quot; in df.columns:

#df.drop([&quot;Volume&quot;], axis = 1, inplace = True)

#if &quot;High&quot; in df.columns:

#df.drop([&quot;High&quot;], axis = 1, inplace = True)

#if &quot;Low&quot; in df.columns:

#df.drop([&quot;Low&quot;], axis = 1, inplace = True)

return df

tickerSymbol = &#39;MSFT&#39;

df = get\_stocks(tickerSymbol, end = &quot;2022-05-10&quot;)

df

df.apply(lambda x: x == 0).sum()

df.isnull().sum()

def fill\_missing(df):

new = df.copy()

idx = pd.date\_range(new.index[0], new.index[-1])

new = new.reindex(idx, fill\_value=0)

n = new.shape[0]

for i in range(1, n):

if new.iloc[i, :].sum() == 0:

new.iloc[i, :] = new.iloc[i-1, :]

return new

df = fill\_missing(df)

def plot\_open(df, tickerSymbol, start = None, end = None):

if not start:

start = df.index[0]

if not end:

end = df.index[-1]

plt.figure(figsize=(24, 10))

plt.title(&quot;Цена открытия акций &quot; + tickerSymbol, fontsize = 20)

plt.plot(df[&quot;Open&quot;][start:end])

plt.grid()

plt.xlabel(&#39;Дата&#39;, fontsize=15)

plt.ylabel(&quot;Цена&quot;, fontsize=15)

plt.show()

years = 5

data\_prep = get\_dataset(df,start=-years\*365, lag\_max = lag\_max)

data\_prep

from keras.models import Sequential

from keras.layers import Dense, Dropout, LSTM

from sklearn.preprocessing import MinMaxScaler

import math

np.random.seed(7)

daf = pd.DataFrame(data\_prep[&quot;Open&quot;].copy())

dataset = daf.values

dataset

dataset = dataset.astype(&#39;float32&#39;)

scaler = MinMaxScaler(feature\_range=(0, 1))

dataset = scaler.fit\_transform(dataset)

train\_size = int(len(dataset) \* 0.959)

test\_size = len(dataset) - train\_size

train, test = dataset[0:train\_size,:], dataset[train\_size:len(dataset),:]

print(len(train), len(test))

def create\_dataset(dataset, look\_back=1):

dataX, dataY = [], []

for i in range(len(dataset)-look\_back-1):

a = dataset[i:(i+look\_back), 0]

dataX.append(a)

dataY.append(dataset[i + look\_back, 0])

return np.array(dataX), np.array(dataY)

look\_back = 1

trainX, trainY = create\_dataset(train, look\_back)

testX, testY = create\_dataset(test, look\_back)

trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

model = Sequential()

model.add(LSTM(4, input\_shape=(1, look\_back)))

model.add(Dense(1))

model.compile(loss=&#39;mean\_squared\_error&#39;, optimizer=&#39;adam&#39;)

model.fit(trainX, trainY, epochs=100, batch\_size=1, verbose=2)

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

# invert predictions

trainPredict = scaler.inverse\_transform(trainPredict)

trainY = scaler.inverse\_transform([trainY])

testPredict = scaler.inverse\_transform(testPredict)

testY = scaler.inverse\_transform([testY])

trainScore = math.sqrt(mse(trainY[0], trainPredict[:,0]))

print(&#39;Train Score: %.2f RMSE&#39; % (trainScore))

testScore = math.sqrt(mse(testY[0], testPredict[:,0]))

print(&#39;Test Score: %.2f RMSE&#39; % (testScore))

r2=r2\_score(testY[0],testPredict[:,0])

print(r2)

trainPredictPlot = np.empty\_like(dataset)

trainPredictPlot[:, :] = np.nan

trainPredictPlot[look\_back:len(trainPredict)+look\_back, :] = trainPredict

# shift test predictions for plotting

testPredictPlot = np.empty\_like(dataset)

testPredictPlot[:, :] = np.nan

testPredictPlot[len(trainPredict)+(look\_back\*2)+1:len(dataset)-1, :] = testPredict

# plot baseline and predictions

plt.plot(scaler.inverse\_transform(dataset))

plt.plot(trainPredictPlot)

plt.plot(testPredictPlot)

plt.show()

Приложение(код) часть 2

!pip install yfinance

!pip install statsmodels

!pip install arch

import pandas as pd

import yfinance as yf

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import csv

from tqdm import tqdm\_notebook as tqdm

from numpy import genfromtxt

from data\_preparation\_lag import \*

from scipy.stats import pearsonr

from sklearn.metrics import r2\_score

def get\_stocks(ticker, start = &quot;1900-01-01&quot;, end = None):

tickerData = yf.Ticker(tickerSymbol)

df = tickerData.history(period=&#39;1d&#39;, start=start, end=end)

if &quot;Stock Splits&quot; in df.columns:

df.drop([&quot;Stock Splits&quot;], axis = 1, inplace = True)

if &quot;Dividends&quot; in df.columns:

df.drop([&quot;Dividends&quot;], axis = 1, inplace = True)

if &quot;Volume&quot; in df.columns:

df.drop([&quot;Volume&quot;], axis = 1, inplace = True)

if &quot;High&quot; in df.columns:

df.drop([&quot;High&quot;], axis = 1, inplace = True)

if &quot;Low&quot; in df.columns:

df.drop([&quot;Low&quot;], axis = 1, inplace = True)

if &quot;Close&quot; in df.columns:

df.drop([&quot;Close&quot;], axis = 1, inplace = True)

return df

tickerSymbol = &#39;MSFT&#39;

df = get\_stocks(tickerSymbol, end = &quot;2022-05-10&quot;)

df

def fill\_missing(df):

new = df.copy()

idx = pd.date\_range(new.index[0], new.index[-1])

new = new.reindex(idx, fill\_value=0)

n = new.shape[0]

for i in range(1, n):

if new.iloc[i, :].sum() == 0:

new.iloc[i, :] = new.iloc[i-1, :]

return new

df=fill\_missing(df)

df

forward = 73

lag\_max = 3

history\_r2 = []

data\_prep = get\_dataset(df,start=-5\*365,lag\_max=lag\_max)

#data\_prep=df.copy()

data\_prep

def smooth(df, ws = 7):

df\_upd = np.copy(df)

for i in range(ws//2, len(df\_upd) - ws//2):

df\_upd[i] = np.median(df\_upd[i-ws//2:i+ws//2])

return df\_upd

def visualization\_result(X\_train,y\_train, X\_test,y\_test, predict, name, forward, ws = None):

if ws:

y\_train = smooth(y\_train, ws)

y\_test = smooth(y\_test, ws)

predict = smooth(predict, ws)

plt.rc(&#39;figure&#39;, figsize=(20, 8))

plt.plot(X\_train[-forward\*5:], y\_train[-forward\*5:], color = &quot;g&quot;, label = &quot;Train&quot;)

plt.plot(X\_test, y\_test, label = &quot;Test&quot;)

plt.plot(X\_test, predict, lw=5, label = &quot;Prediction&quot;)

plt.grid()

plt.title(f&quot;{name}&quot;, fontsize = 20)

plt.xlabel(&#39;Дата&#39;, fontsize=15)

plt.ylabel(&quot;Цена&quot;, fontsize=15)

plt.legend(prop={&#39;size&#39;: 20})

plt.show()

import itertools

from sklearn.metrics import mean\_squared\_error as mse

def all\_comb(params):

keys = params.keys()

values = (params[key] for key in keys)

combinations = [dict(zip(keys, combination)) for combination in itertools.product(\*values)]

return combinations

def Greed\_Search(model, X\_train, y\_train, X\_test, y\_test, lag\_max, params):

best\_mape = float(&quot;inf&quot;)

best\_combo = {}

combs = all\_comb(params)

for comb in tqdm(combs):

CB = TimeModel(model = model , look\_back= lag\_max, \*\*comb)

CB.fit(X\_train, y\_train)

y\_pred = CB.forecast(X\_test)

result = mse(y\_test, y\_pred)

if result &lt; best\_mape:

best\_mape = result

best\_combo = comb

return best\_mape, best\_combo

X\_train, y\_train, X\_test, y\_test = get\_train\_test(data\_prep, forward)

train\_ind, test\_ind = data\_prep.index[:-forward], data\_prep.index[-forward:]

from sklearn.svm import SVR

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import StandardScaler

sv = SVR(C=100,gamma=0.001,kernel=&#39;rbf&#39;)

sv.fit(X\_train, y\_train)

y\_pred = sv.predict(X\_test)

r2 = r2\_score(y\_test, y\_pred)

print(f&quot;R^2 score(coefficient of determination): {r2} &quot;)

history\_r2.append(r2)

visualization\_result(train\_ind, y\_train, test\_ind, y\_test, y\_pred, &quot;SVR&quot;, forward, 4)

from sklearn.neighbors import KNeighborsRegressor

KN = KNeighborsRegressor(n\_neighbors=6,p=3,weights=&#39;uniform&#39;)

KN.fit(X\_train, y\_train)

y\_pred = KN.predict(X\_test)

r2 = r2\_score(y\_test, y\_pred)

print(f&quot;R^2 score(coefficient of determination): {r2} &quot;)

history\_r2.append(r2)

visualization\_result(train\_ind, y\_train, test\_ind, y\_test, y\_pred, &quot;KNeighborsRegressor&quot;, forward, 4)

Приложение(часть 3) data\_preparation

import numpy as np

import pandas as pd

def get\_dataset(dataset, start=None, end=None, lag\_max=30):

&quot;&quot;&quot;

Based on the ordinary data from the field, we generate a lot of statistical

and lag features and return a dataset with a very large number of features.

:param dataset: data with stocks. (pd.DataFrame)

:param start: the point from which the output dataset starts to be generated. (int)

:param end: the point to which the output dataset will be generated. (int)

:param lag\_max: how much lags it will be generated.

:return: a dataset in which all the necessary features have already been added

and it remains only to configure training and validation on it. (pd.DataFrame)

&quot;&quot;&quot;

if not start:

start = 0

if not end:

end = dataset.shape[0]

#data\_prep = pd.DataFrame(dataset[&quot;Open&quot;])

data\_prep=dataset.copy()

if end == 0:

data\_prep = data\_prep.iloc[start:]

else:

data\_prep = data\_prep.iloc[start:end]

if lag\_max &gt; 2:

for lag\_num in range(1, lag\_max, 1):

data\_prep[&quot;Open\_lag\_{}&quot;.format(lag\_num)] = data\_prep[&#39;Open&#39;].shift(lag\_num)

#data\_prep[&#39;year&#39;] = data\_prep.index.year

#new\_features = pd.get\_dummies(data\_prep.index.year)

#new\_features.columns = list(map(lambda x: &#39;year\_&#39; + str(x), new\_features.columns))

#new\_features.index = data\_prep.index

#data\_prep = pd.concat((data\_prep, new\_features), axis=1)

#new\_features = pd.get\_dummies(data\_prep.index.month)

#new\_features.columns = list(map(lambda x: &#39;month\_&#39; + str(x), new\_features.columns))

#new\_features.index = data\_prep.index

#data\_prep = pd.concat((data\_prep, new\_features), axis=1)

#new\_features = pd.get\_dummies(data\_prep.index.week)

#new\_features.columns = list(map(lambda x: &#39;week\_&#39; + str(x), new\_features.columns))

#new\_features.index = data\_prep.index

#data\_prep = pd.concat((data\_prep, new\_features), axis=1)

#new\_features = pd.get\_dummies(data\_prep.index.weekday)

#new\_features.columns = list(map(lambda x: &#39;dow\_&#39; + str(x), new\_features.columns))

#new\_features.index = data\_prep.index

#data\_prep = pd.concat((data\_prep, new\_features), axis=1)

#new\_features = pd.get\_dummies(data\_prep.index.hour)

#new\_features.columns = list(map(lambda x: &#39;h\_&#39; + str(x), new\_features.columns))

#new\_features.index = data\_prep.index

#data\_prep = pd.concat((data\_prep, new\_features), axis=1)

#data\_prep[&#39;day&#39;] = data\_prep.index.day

data\_prep.dropna(inplace=True)

return data\_prep

def get\_train\_test(prepared\_dataset, th=90):

&quot;&quot;&quot;

Split dataset in train and test parts.

:param prepared\_dataset: a dataset in which all the necessary features have already been

added and it remains only to configure training and validation on it. (pd.DataFrame)

:param th: threshold for allocating data for train and test. (int)

:return: two pairs, where arguments and targets are located within the pair,

the fraction corresponds to the &#39;th&#39; parameter. (4x np.array)

&quot;&quot;&quot;

X\_train = prepared\_dataset.drop([&#39;Open&#39;], axis=1).values[:-th]

y\_train = prepared\_dataset[&#39;Open&#39;].values[:-th]

X\_test = prepared\_dataset.drop([&#39;Open&#39;], axis=1).values[-th:]

y\_test = prepared\_dataset[&#39;Open&#39;].values[-th:]

return X\_train, y\_train, X\_test, y\_test

def mape(y\_true, y\_pred):

&quot;&quot;&quot;

Mean absolute percentage error.

:param y\_true: ground truth (correct) target values. (one dim array)

:param y\_pred: estimated target values. (one dim array)

:return: the value of mape score between y\_true and y\_prediction.

&quot;&quot;&quot;

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

assert y\_true.shape == y\_pred.shape

return np.mean(np.abs((y\_true - y\_pred) / y\_true))