ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему: Торговая система для крипто-бирж

(промежуточный, этап 2)

Выполнили:

Студент группы БПМИ209		Л.В.Прокопчук
	Подпись	И.О.Фамилия
Студент группы БПМИ209		Л.И.Рыбаков
	Подпись	И.О.Фамилия
Студент группы БПМИ209		И.Ю.Бондырев
	Подпись	И.О.Фамилия

17.02.2022
Дата

Принял:

Руководитель проекта Казаков Евгений Александрович
Имя, Отчество, Фамилия

разработчик
Должность, ученое звание

Facebook inc.
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки 17.02 2022 10
Оценка (по 10-ти бальной шкале) Подпись

Москва 2022

Содержание

Аннотация

Исследование и написание торговых стратегия для децентрализованных бирж на Layer-2[1].

[1] Технологии масштабирования, позволяющие увеличить скорость и пропускную способность блокчейна. Обычно это делается за счет децентрализованности. Самые распространенные разновидности — rollups (свертки, собирают несколько транзакций в одну и записывают в блокчейн) и sidechain (блокчейн, который опирается на масштабируемую сеть. Часто используют менее децентрализованную и, как следствие, более быстрые консенсусные алгоритмы)

1 Введение

1.1 Актуальность проблемы

Сегодня, кого не спроси, все знают, что такое биткоин, или, по крайней мере, говорят, что знают. Разговоры о криптовалютах и их производных в мире не утихают уже несколько лет, но, к сожалению, большинство из них очень поверхностные и не доходят даже до обсуждения принципа работы блокчейна в общих словах. Несмотря на кажущуюся сложность устройства, 2 самых больших блокчейна (Bitcoin и Ethereum) критически не справляются с нагрузкой, возложенной на них желающими воспользоваться их плюсами. Из-за того, что сеть Эфириума может обрабатывать лишь 15 транзакций в секунду, комиссия, которую нужно заплатить, чтобы транзакция была одной из этих 15 доходит до $70, что делает любой токен на блокчейне непригодным для использования в качестве обычной фиатной валюты. Чтобы снизить нагрузку на мэйннет[2], были разработаны и все еще разрабатываются несколько альтернативных решений, которые одним словом называются Layer-2 решения. Это надстройки над блокчейном, которые увеличивают пропускную способность и скорость в ущерб децентрализованности. Мы считаем, что пока не придумали более изящного способа достичь тех же результатов, которые дают L2 решения, данная технология будет развиваться, а актуальности нашей темы будет расти.

1.2 Цели и задачи

1.2.1 Цель

Написать трэйдинг систему, которая сможет стабильно выходить в плюс.

1.2.2 Задачи

- Проведение исследований по стратегиям трэйдинга.
- Проверка работоспособность стратегий.
- Создание инфраструктуры, позволяющей взаимодействовать с биржей автоматизированно.
- Сбор данных и обучение модели.
- Написание программы, совершающей сделки.
- Обзор и сравнительный анализ источников и аналогов

К сожалению, выбранная нами тема мало освещается в источниках любого вида: никто не захочет делиться прибыльной стратегией. Многое нам приходилось и придется делать с нуля.

1.3 Статьи

Тем не менее, существуют статьи, описывающие некоторые возможные подходы к написанию алгоритмов HFT. Например, есть ресурс [8], на котором описывается стратегия маркет-мейкинга, аналог которой мы попытались реализовать. Но материалы такого рода, находящиеся в открытом доступе, с течением времени теряют свою актуальность: если большое количество участников рынка придерживается одной схемы действий, то вскоре она перестает приносить прибыль. По этой причине мы старались не ориентироваться на подобные источники.

1.4 Документация бирж

Основным же источником информации для нас служила документация API [3] [1] бирж, к которым мы подключались. С помощью нее был написан коннектор, инкапсулирующий процесс подключения и взаимодействия с биржей, произведен сбор необходимой информации: список сделок за последний месяц, состояние о счете и т.п.

[2]Основная сеть блокчейна, на которой криптовалюта имеет реальную стоимость. Есть также сети для тестирования разработок. На них валюту можно получить по запросу от специальных адресов

1.5 Документация библиотек

Для машинного обучения мы использовали CatBoost [2] — это библиотека от Яндекса для градиентного бустинга, надстройки над решающими деревьями. КэтБуст для нас лучшее решение, потому что это самая быстрая библиотека для классификации среди аналогов и проста в использовании.

1.6 Аналоги

На крипто валютном рынке существует множество торговых ботов, но информации об их характеристиках и принципах работы практически нет. Мы можем судить об их доходности, лишь по каким-то сомнительным заявлениям или косвенным признакам. В открытом доступе в основном находятся боты, которые предоставляют лишь интерфейс взаимодействия с биржей [7] [5]: "ручная" покупка и продажа токенов, выставление лимитных ордеров и т.п.. Такие решения не представляют для нас никакого интереса.

2 Market Making

2.1 Что это такое

Когда человек приходит на биржу, он хочет купить актив по рыночной цене. Если биржа A продает биткоины по 11k\$, а биржа B по 10k\$, то выгоднее покупать биткоины у B. Чтобы A не терять клиентов, она пользуется услугами маркет-мейкеров.

Маркет-мейкеры решают проблему дисбаланса цен между биржами. Они могут за вознаграждение от биржи A продать у них биткоины и выровнять курс до 10k\$, и тогда всем остальным будет снова выгодно торговать на бирже A. Аналогично работает, когда на бирже A курс ниже относительно других бирж: маркет-мейкеры закупятся биткоинами. В стабильное время маркет-мейкеры занимаются поддержанием маленького спреда, то есть делают так, что цены покупки и продажи отличаются как можно меньше.

На децентрализованных биржах все еще интереснее. Так как там биткоин не привязан к бирже, то стратегия выше выгодня для маркет-мейкеров, потому что они могут на бирже A продать биткоины по 11k\$, и купить на B по 10k\$, заработав разнице в ценах, пока они не выровняются.

Однако межбиржевые операции очень дорогие и долгие, поэтому чаще всего маркет-мейкеры зарабатывают на резких инертных скачках рынка и вознаграждения от биржи.

Для второго нужно очень много денег, так что попробуем заработать на первом.

2.2 Как это работает

Заработать можно в предположении краткосрочно высоко-инерционного рынка.

Определение 1. Пробитием будем называть ту заявку, которую мы не успели отменить, и по ней у нас открылась позиция.

Алгоритм 2. *Market-Making orders*

1. *Выставляем лимитные заявки в обе стороны на* $\pm\Delta_1$ *от индекс-цены.*
2. *Когда индекс-цена изменяется, переставляем заявки на ту же самую* $\pm\Delta_1$*, но уже от новой цены.*
3. *Если переставиться успели, и нас не пробили, то переходим к шагу 2*
4. *Неумаляя общности нас пробили на покупку по цене* p*. Отменяем все заявки*
5. *Выставляем заявку на продажу на* Δ_2 *от цены, по которой нас пробили на покупку*
6. *Когда заявка исполнилась, возвращаемся к шагу 1*

В итоге на шаге 6 мы заработает $\Delta_2 \cdot p$.

2.3 Реализация

Стратегия в репозитории

Программа работает в трех потоках, которые нужны для:

1. Выставление ордеров
   (a) В этом потоке мы получаем новый трейды.
   (b) Если цена нового трейда не такая, как у прошлого, и цена ордера, который надо выставить не такая, как у нас сейчас стоит, то мы отменяем текущие ордера и выставляем новые.
   (c) Перестановка ордеров переходит не через сначала отмену, старых, а потом выставления новых ордеров, а непосредственно через переставление позиций аргументом `cancel_id` в функции `create_order`. Это позволило сократить количество запросов в два раза, а значит ошибка `Too many requests` будет встречаться реже.
2. Получение обновлений ордербука
   (a) В предыдущем потоке во время выставления и отмены ордеров новые обновления по вебсокету не приходят, но обновления ордер бука нам нужны, потому что по нему мы определяем цену оредров.
   (b) Поэтому получения ордер бука вынесено в отдельный поток, чтобы всегда именть свежий стакан.
3. Проверки положение ордеров
   (a) Бывает так, что новые трейды не приходят, но окно спреда ордер бука меняется.
   (b) Нам важно наше окно трейдов держать строго на $\pm$ какой-то спред вокруг бидсов и асков.
   (c) Поэтому мы раз в какое-то время проверяем этот баланс, и если происходит дисбаланс, то мы обновляем наши ордера.

2.4 Результаты

Мы торговали на бирже `DyDx`. Она входит в топ самых популярных децентрализованных бирж. Количество сделок в месяц в ней около 30 тысяч и по биткоинку, и по эфиру. На самом деле это очень мало: в среднем 0.3 сделки в секунду, то есть одна сделка в 3-4 секунды. Соответственно рынок совсем не высоко-инерционный, поэтому после того, как нас пробили в одну сторону, до пробития во вторую, может пройти несколько десятков минут. Это превышает наш таймаут, потому что если мы слишком долго будем держать открытую позицию, то повышается вероятность, что рынок пойдет в обратную сторону и мы потеряем еще больше денег.

Еще часто бывали случаи, когда рынок отскакивает в нужную нам сторону, но на величину меньше комиссии, соответственно, мы теряем деньги.

2.5 Выводы

Если бы у нас была нулевая или близка к нулевой комиссия, то мы бы были в плюсе. Но с той, что у нас есть, в минусе.

3 Индикаторы

Для построения модели в катбусте мы использовали индикаторы, которые зависят от трэйдов в окне. Класс `Indicators` содержит функции, заполняющие значения фичей и столбца таргета: `fill_features_values` и `fill_target_values`.

3.1 `fill_target_values`

Функция в репозитории

Функция принимает словарь для заполнения значений, два окна трэйдов и параметры ордеров `stop_profit` и `stop_loss`. Функция считает, исполнились бы эти ордеры и выставляет соответствующее значение в словарь для заполнения.

3.2 `fill_features_values`

Функция в репозитории

Предложение 3. *Окно длины t – окно, в котором есть все трэйды, случившиеся за интервал времени длины t секунд.*

Функция принимает словарь для заполнения, окно трэйдов, и два списка вариантов числовых параметров фичей n и t. Функция заполняет словарь значениями фичей, которые представлены ниже:

- `seconds_since_midnight` – количество секунд с начала дня.
- `seconds_since_n_trades_ago` – количество секунд, прошедших с первого трэйда в окне из n последних трэйдов.
- `WI_exp_moving_average` – экспоненциальное среднее в окне длины t.
- `WI_weighted_moving_average` – взвешанное среднее в окне длины t.
- `WI_trade_amount` – количество сделок в окне длины t.
- `WI_trade_volume` – суммарный объем сделок в окне длины t.
- `WI_open_close_diff` – разница между ценой открытия и ценой закрытия в окне длины t.
- `WI_stochastic_oscillator` – стохастический осцилятор. Рассчитывается по формуле $\frac{\texttt{CUR}-\texttt{MIN}}{\texttt{MAX}-\texttt{MIN}}$, где `CUR` - цена последней сделки, а `MIN` и `MAX` - наименьшая и наибольшая цена сделки соответственно в окне длины t.

Вторая фича зависит от значения n и от направления трэйдов в окне, поэтому для каждой комбинации направления и параметра n в таблице будет свой столбец.

Последние 5 фичей зависят от t и от направления трэйдов в окне, поэтому для каждой комбинации направления, параметра t и фичи в таблице будет свой столбец.

4 Трейдер

Трейдер в репозитории

Определение 4. Трейдер – программа, которая на основе готовой стратегии, совершает сделки.

4.1 Использованные технологии

4.1.1 `Асинхронная функция`

Определение 5. Асинхронная функция – функция, которая не блокирует основой поток программы.

Наш проект написан на питоне, а он одно-поточный. Поэтому нужно было пользоваться всеми возможностями асинхронности. Для этого в питоне есть очень простой интерфейс:

```
async def slow_calculation():
    return ...

async def hard_calculation():
    return ...

def main():
    loop = asyncio.get_event_loop()
    loop.create_task(
        slow_calculation(), name="slow_calculation"
    )
    loop.create_task(
        hard_calculation(), name="hard_calculation"
    )
    loop.run_forever()
```

Здесь мы создали две задачи: `slow_calculation` и `hard_calculation`, которые будут выполняться асинхронно, то есть не мешать друг другу своим вычислениями.

4.1.2 `asyncio.Event`

В трейдере, помимо асинхронных функций, используется класс `asyncio.Event` - событие, позволяющий ждать задачам, пока событие не произойдет в другой задаче и элемент класса не будет разблокирован.

```
async def waiter(event):
    print('waiting for it ...')
    await event.wait()
    print('... got it!')

async def main():
    # Create an Event object.
    event = asyncio.Event()

    # Spawn a Task to wait until 'event' is set.
    waiter_task = asyncio.create_task(waiter(event))

    # Sleep for 1 second and set the event.
    await asyncio.sleep(1)
    event.set()

    # Wait until the waiter task is finished.
    await waiter_task

asyncio.run(main())
```

В данном примере функция `waiter` не завершится до тех пор, пока функция `main` не разблокирует событие строчкой `await asyncio.sleep(1)`, потому что `waiter` находится в ожидании после строчки `await event.wait()`.

С помощью этого класса можно синхронизировать работу алгоритма торговли с ответами от биржи.

4.2 Устройство трейдера

Используемые параметры:

- `trailing_percent` – процент, на который цена stop-loss ордера отстает от рыночной при отправке trailing-stop ордера.
- `profit_threshold` – процент, на который цена лимитки, выставленной по take-profit ордеру отличается от рыночной.
- `sec_to_wait` – количество секунд, через которое цикл торговли считается завершенным, если ни take-profit, ни trailing-stop ордеры не были исполнены.

Трейдер состоит из двух асинхронных задач и вспомогательных функций.

Из ключевых вспомогательных функций нужно отметить функцию `reset`, которая переводит все поля класса в состояние для ожидания скачка на бинансе. Она вызывается после успешного завершения цикла торговли, или после отмены ордеров и закрытия позиции в случае, когда какой либо ордер не смог исполниться.

Основные асинхронные задачи:

- `listen_binance`
- `account_listener`

Пройдемся по каждой

4.2.1 `listen_binance`

Функция в репозитории

Определение 6. Слушать [что-то] – значит подключиться по веб-сокету к какому-то серверу и получать от него обновления.

Эта функция реализует логику торговли. В этом трейдере по техническим причинам не получилось использовать функционал коннектора для бинанса, поэтому подписываться на обновления пришлось в ручную. Функция держит окно трейдов на одном направлении фиксированной длины, и если перепад цены превысил порог, то запускается алгоритм торговли:

1. Отправляется маркет ордер. Функция входит в режим ожидания с помощью `asyncio.Event` и ждет, пока с биржи придет ответ об исполнении ордера. Если с биржи приходит сообщение об отмене ордера, то вызывается функция `reset` и трэйдер возвращается в состояние ожидания скачка цены на бинансе.
2. Когда `account_listener` сообщил об успешном выполнении маркет ордера и сохранил цену по которой он исполнился, трейдер выставляет `take-profit` и `trailing-stop` ордеры с заданными при инициализации класса параметрами и на протяжении `sec_to_wait` секунд ожидает исполнения хотя бы одного из них.
   - Если за `sec_to_wait` секунд ордеры не закрылись, то они отменяются. Далее закрывается позиция в которую мы зашли маркетом, и когда с биржи приходит сообщение о закрытии позиции, цикл торговли считается завершенным.
   - Если какой-то из ордеров исполнился, то мы в любом случае отменяем оставшийся. Если исполнился лимит, то мы в плюсе, если трэйлинг стоп, то, скорее всего, в минусе. После отмены цикл считается завершенным.

4.2.2 `account_listener`

Функция в репозитории

Эта задача слушает изменения в наших ордерах и позициях, обновляет цену, по которой исполнился маркет, выставляет флаги и разблокирует нужные события, на которых спит функция `listen_binance`, тем самым обеспечивая её синзронизацию с биржей.

5 Описание функциональных требований к программному проекту

5.1 Сбор данных

Нужно обеспечить удобный механизм сбора исторических данных с бирж, на которых будет тестироваться и обучаться система.

5.2 Измерение скорости соединения

Так как счет идет на миллисекунды, мы всегда должны иметь четкое представления, какое время у нас займет отправка и получения пакета данных. Для этого должен быть предусмотрен отдельный модуль, который будет замерять скорость соединения с различными сервисами.

5.3 Коннектор к бирже

В программе должны быть коннекторы к биржам. Это класс, в конструктор которого подаются приватные ключи кошельков. После этого можно работать с биржей: смотреть информацию о счете, валютаю, отправлять и отменять ордера. Надо реализовать функционал, который предоставляет апи, чтобы можно было его использовать в трейдинг-стратегиях.

5.4 Машинное обучение

Индикаторов, по которым можно строить прогнозы, очень много, поэтому ручными методами не получится подобрать правильное соотношение весов. Здесь нужно машинное обучение, которое принимает на вход пред обработанные данные сделок, а на выход выдает модель, которую можно использовать в трейдинг-стратегии. Важно, чтобы модель была устойчива к тому, что в датасет добавляется или убирается индикаторы. Примеры использования

5.5 Коннектор

Предоставляем класс, который способен взаимодействовать с биржами по API, например:

- Отправка ордеров

```
connector.send_order(
        symbol=ETH_USD, side=BUY, price=1, quantity=0.1
)
```

- Получение текущих позиций

```
connector.get_our_positions(
        opened=True, symbol=ETH_USD
)
```

- Получение трейдов за определенный промежуток времени

```
connector.get_historical_trades(
        market=BTC_USD,
        begin="2021-12-12 09:00:00",
        end="2021-12-12 12:00:00"
)
```

- Информация о конкретном рынке

```
connector.get_symbol_info(market=ETH_USD)
```

- Измерение скорости

```
speed_measure = SpeedMeasure(connector)
speed_measure.get_connector_funcs_exec_times(
        market=ETH_USD,
        side=BUY,
        iters_num=10,
        filename="connector_funcs_exec_times.json",
    )
```

- Измерение задержки до биржи

```
speed_measure.get_orders_processing_delays(
        market=ETH_USD,
        side=BUY,
        orders_num=10,
        filename="orders_processing_delays.json",
)
```

6 Описание нефункциональных требований к программному проекту

6.1 Безопасность

Финансы — чувствительная тема, поэтому наша программа не должна допускать утечек данных о кошельках и приватных ключах. Нужно обеспечить безопасное хранение.

6.2 Отказоустойчивость

Во время трейдинга торговая система получает сотни обновлений от разных бирж, их обрабатывает, строит прогнозы и торгует. Нужно сделать так, чтобы система была готова к большим нагрузкам, и поведение было однозначно определено. Еще нужно проработать быстрое отключение торговой системы от торгов, если ее поведение станет неадекватным, и можно было бы быстро закрыть открытые заявки.

6.3 Скорость

В трейдинге важна каждая миллисекунда, поэтому цель — минимизировать время обработки, отправки и принятия данных.

6.4 Переиспользование кода

Нужно выстроить архитектуру проекта так, чтобы можно было быстро и легко тестировать свои гипотезы, поэтому код, который есть в проекте, должен быть написан так, чтобы его можно было легко понять и переиспользовать в других местах.

6.5 Масштабируемость

Система должна быть расширяема на несколько бирж и потенциально работать с большим количеством предсказательных моделей.

7 Дополнительный результаты

Помимо задач, которые нам поставил руководитель, мы сделали еще:

7.1 Визуализация

Перед тем, как работать с данными, надо понять, как они устроены: попарное распределение классов, плотность каждого из индикаторов. Мы все это сделали. Теперь стало гораздо проще подбирать параметры для модели машинного обучения.

7.2 Маркет-Мэйкинг стратегия

Это стратегия, когда мы держим окно из зеркальных заявок на каком-то уровне от индекс-цены. Утверждается, что если нас пробили с одной стороны, то почти сразу пробьют и со второй стороны, и мы окажемся в плюсе. Но все оказалось не так: при пробитии нас с одной стороны, рынок продолжал идти в ту же сторону.

7.3 Контролирующий бот

Мы сделали телеграм бота, на которого можно по паролю подписаться и получать обновления состояния аккаунта на бирже. Это полезно, когда ты запускаешь торговую стратегию, куда-то отходишь, но при этом всегда можешь контролировать, что с ней происходит, через телеграм.

7.4 CI/CD, тесты, линтер

Любая ошибка в торговой системе может потерять наши деньги, поэтому важно, чтобы весь код всегда был рабочим. Для этого мы все, что смогли, обложили тестами с использованием утилиты PyTest [11]. Теперь при каждом пулл реквесте у нас запускается тестирующая система, и если какие-то тесты не проходят, то мы запрещаем дальнейший пуш. Реализовали мы это через GitHub Actions [6].

Еще нам хочется, чтобы весь код был консистеным. Для этого мы используем линтер Black [9] и статический анализатор PyLint [10]. В совокупности эти утилиты поддерживают консистентность нашего кода и могут еще до запуска теста, выдать синтаксические ошибки.

7.5 Смарт контракты

С помощью смарт контрактов можно занимать у других людей миллиарды, трейдить на них, и потом возвращать криптовалюту обратно. Звучит заманчиво. Мы написали смарт конракты на Solidity [12], и залили их в сеть эфира через Brownie [4].

Список литературы

[1] Binance. Binance documentation. 2022. URL: https://binance-docs.github.io/apidocs/spot/en/#change-log.

[2] Catboost. Catboost documentation. 2022. URL: https://catboost.ai/en/docs/.

[3] DydxProtocol. Dydx documentation. 2022. URL: https://docs.dydx.exchange/#general.

[4] ETHBrownie. Brownie. 2022. URL: https://github.com/eth-brownie/brownie.

[5] FreqTrade. Free, open source crypto trading bot, 2022. URL: https://github.com/freqtrade/freqtrade.

[6] GitHub. Github actions. 2022. URL: https://docs.github.com/en/actions.

[7] Haehnchen. Cryptocurrency trading bot in javascript. 2022. URL: https://github.com/Haehnchen/crypto-trading-bot.

[8] HFTbattle. Hft strategy. 2022. URL: https://docs.hftbattle.com/ru/strategy/advanced.html.

[9] PFS. Black. 2022. URL: https://github.com/psf/black.

[10] PyCQA. Pylint. 2022. URL: https://github.com/PyCQA/pylint.

[11] PytestDev. Pytest. 2022. URL: https://github.com/pytest-dev/pytest.

[12] Solidity. Solidity. 2022. URL: https://docs.soliditylang.org/en/v0.8.12/.