

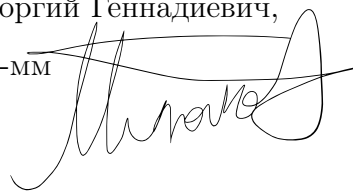
Санкт-Петербургский государственный университет  
Прикладная математика и информатика

Отчёт по учебной практике 4 (научно-исследовательской работе) (семестр 7)


Применение стохастических методов в машинном обучении для  
решения финансовых задач

Выполнил:

Милюшков Георгий Геннадиевич,  
группа 20.Б04-мм



Работа выполнена  
в удовлетворительном  
объеме



Научный руководитель:

к.ф.-м.н., доцент Шпилев Петр Валерьевич

Кафедра статистического моделирования

Отметка о зачете:

<< Работа выполнена на удовлетворительном уровне и может быть зачтена с оценкой С.>>

Санкт-Петербург

2023

Отзыв на учебную практику (научно – исследовательскую работу) студента  
4-го курса бакалавриата Милюшкова Георгия Геннадиевича.

Работа посвящена изучению методов машинного обучения с подкреплением на основе стохастической оптимизации в аспекте их применения к решению прикладных задач из области финансов. В качестве примера рассматривается задача создания биржевого трейдера, способного обучаться на основе рыночных переменных, совершать в соответствии с текущей ситуацией некоторые действия (покупать, продавать и ждать), а также, оценивать собственную эффективность. В процессе исследования студент самостоятельно ознакомился с основными подходами, основанными на использовании стохастической оптимизации, в частности, с Q-обучением, методами Монте-Карло, методом временных различий и SARSA. В работе представлена реализация метода Q-обучения на языке Python и проведен анализ его эффективности.

Работа написана аккуратно, поставленная задача реализована в достаточно полном объеме. Считаю, что работа может быть зачтена с оценкой С.



07.01.24

Петр Валерьевич Шпилев

## Оглавление

Введение . . . . .	3
Данные . . . . .	3
Машинное обучение с подкреплением . . . . .	4
Q-обучение . . . . .	5
Биржевой трейдер . . . . .	9
Заключение . . . . .	12
<b>Список литературы . . . . .</b>	<b>14</b>

## Введение

Целью научно-исследовательской работы в 7-м семестре являлось изучение основ машинного обучения с подкреплением и методов стохастической оптимизации. Настоящая работа является началом для работы по применению различных стохастических методов машинного обучения с подкреплением для решения финансовых задач. В рамках этой темы был изучен метод Q-обучения — один из наиболее известных методов машинного обучения с подкреплением. Реализация выполнена на языке программирования Python. Полный код и данные доступны в репозитории GitHub[2].

## Постановка задачи

Задача на 7 семестр научно-исследовательской работы заключалась в изучении основ машинного обучения с подкреплением и началом изучения методов стохастических оптимизаций. В данной работе рассматривается метод Q-обучения для создания биржевого трейдера, способного извлекать уроки из рыночных переменных, генерировать действия (покупка, продажа, ожидание) и оценивать собственную эффективность. Акцент работы делается на подходы, основанные на использовании стохастической оптимизации для настройки параметров агента, чтобы он мог адаптироваться к стохастическим условиям среды и выбирать оптимальные действия, при влиянии случайности. Помимо изучаемого Q-обучения, существуют другие методы стохастической оптимизации, такие как: Методы Монте-Карло, методы временных различий, и SARSA. Данные методы позволяют агенту оценивать и улучшать свою стратегию в условиях неопределенности.

## Данные

Данные которые используются для создания биржевого трейдера, основываются на исторических данных акций компании Apple(APPL) на бирже NASDAQ с 01.01.2014, по 01.01.2018. 80% данных были использованы, как тренировочный набор данных. В насто-

ящем наборе данных об акциях компании Apple имеется информация о дате, наивысшей цене акции в этот день, наименьшей цене акции в этот день, цена на момент открытия биржи NASDAQ, цена на момент закрытия биржи NASDAQ, объем — количество акций, которые торгуются в течении дня и скорректированная цена закрытия — цена после корректировок, таких как выплаты дивидендов и дробление акций.

## Машинное обучение с подкреплением

Обучение с подкреплением (Reinforcement Learning) — область машинного обучения, в которой обучение осуществляется посредством взаимодействия с окружающей средой. В отличие от других парадигм машинного обучения, машинное обучение с подкреплением работает методом проб и ошибок. Агент, окружающая среда, действия, награда, состояние и цель являются ключевыми понятиями машинного обучения с подкреплением, которые являются важными для понимания основ обучения с подкреплением и Q-обучение.

Агент принимает решения и взаимодействует с окружением. Агент стремится максимизировать суммарную награду, получаемую из окружения. В финансовых задачах это может быть торговый алгоритм или система управления портфелем. Агент принимает решения о том, когда покупать или продавать акции на основе текущей информации и своей стратегии.



Окружающая среда, это пространство в котором действует агент. Окружающая среда предоставляет агенту обратную связь в виде награды в ответ на предпринятые им действия. В финансовых задачах рыночные условия, такие как изменения цен на акции, объемы торгов, новости и события, представляют собой составляющие окружающей среды.

Действия это шаги или решения, которые агент может предпринимать. Выбор действия определяется стратегией агента. В настоящей задаче это покупка, продажа или удержание акций. Агент выбирает определенное действие на основе текущей информации и своей стратегии, стремясь максимизировать свою прибыль или минимизировать риски.

Награда это числовая обратная связь, предоставляемая окружением агенту. Награда служит сигналом, указывающим агенту, насколько хорошо он справляется с текущей задачей. Данная функция является часто неоднозначной, так как непонятно какую величину давать и когда. В финансовых задачах это может быть связана с прибылью или убытками от сделки. Например, положительная награда может быть предоставлена за прибыльную сделку, а отрицательная — за убыточную, где размер награды соответствует прибыли или убытку. Процесс обучения с подкреплением включает в себя многократное взаимодействие агента с окружением, в процессе которого агент адаптирует свою стратегию, чтобы максимизировать получаемую награду. Для этого используются различные алгоритмы обучения, такие как Q-обучение, методы глубокого обучения (Deep Q Networks, DQN), алгоритмы стратегий (Policy Gradient), и другие.

Состояние представляет собой информацию, которая описывает текущее положение и условия агента в среде. Состояние содержит всю необходимую информацию для принятия решения агентом о том, какое действие предпринять. Состояния могут быть обширными и включать в себя различные параметры. В финансовых задачах это могут быть различные технические индикаторы, время, цены и положение рынка. Цель это то что агент стремится достичь, например максимальную прибыль.

Важным понятием является также политика. Оно определяет поведение агента в окружающей среде. Способ выбора агентом действия, которое он будет выполнять, зависит от политики. В задаче создания биржевого трейдера с использованием Q-обучения, политикой к примеру может быть покупка акций, если индикаторы технического анализа указывают на переход восходящего тренда и продажа, если указывают на переход нисходящего тренда.

## Q-обучение

Q-обучение является фундаментальным для разработки алгоритмов, способных принимать решения в условиях неопределенности и максимизировать кумулятивное вознаграждение в долгосрочной перспективе. Q-обучение является формой обучения с подкреплением, где агент, принимая решения в среде, стремится оптимизировать Q-функцию, оценивающую ценность различных действий в конкретных состояниях. Этот метод позволяет агенту учитывать долгосрочные последствия своих действий, что является ключевым в обучении с подкреплением.

### Математическая модель

Машинное обучение с подкреплением в значительной степени основывается на марковских процессах принятия решений. Марковский процесс принятия решений является математической моделью для описания взаимодействия между агентом и окружающей средой, где агент принимает решения, чтобы максимизировать свою кумулятивную награду. Оно задается множеством состояний  $\mathcal{S}$  и множеством действий  $\mathcal{A}$ . Вероятность перехода из состояния  $s$  в  $s'$  при условии действия  $a$  в момент времени  $t$  задается:

$P_a(s, s') = P(\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s, \mathcal{A}_t = a)$ . Награда представляется в виде функции из состояния  $s$  в  $s'$  при условии действия  $a$ , как  $R_a(s, s')$ . Политика является функцией  $\pi : \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ . Она определяется:  $\pi(a, s) = P(\mathcal{A}_t = a | \mathcal{S}_t = s)$ .

### Функция ценности и Q-функция

При обучении агента в машинном обучении с подкреплением фундаментальным является оценивание состояния. В настоящей задаче создание биржевого трейдера важно, чтобы агент мог принимать решения о том, когда и какие торговые действия следует предпринять в зависимости от текущего состояния рынка. Для этого используется функция ценности и Q-функция.

Функция ценности указывает, насколько хорошо для агента пребывание в конкретном состоянии при политике  $\pi$ , то есть оценивает ценность состояния при следовании политике. Данная функция показывает ожидаемую награду, начинающийся со состояния  $s$ , в соответствии с политикой  $\pi$ . Функция ценности задается в виде математического ожидания от будущих наград:

$$V_{\pi}(s) = \mathbb{E}_{\pi}[G|S_0 = s] = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s \right].$$

$\gamma$  — дисконтирующий коэффициент, который определяет относительную важность будущих и немедленных наград. Его значение лежит в диапазоне от 0 до 1. Дисконтирующий коэффициент 0 означает, что немедленные награды более важны, а дисконтирующий коэффициент 1 означает, что будущие награды важнее немедленных. С коэффициентом 0 никакого обучения не будет, поскольку учитываться будут только немедленные награды, с другой стороны, дисконтирующий коэффициент 1 будет неограниченно стремиться к будущим наградам, что может завести в бесконечность. Таким образом, оптимальное значение дисконтирующего коэффициента лежит в диапазоне от 0.2 до 0.8. В финансовых задачах данный коэффициент особенно важен учитывая принцип «Доллар сегодня стоит дороже, чем доллар завтра».

Q-функция является основой Q-обучения. Функция ценности состояния и действия называется Q-функцией. Она показывает, насколько хорошо для агента выполнять конкретное действие в соответствии с политикой  $\pi$ . Q-функция, в отличие от функции ценности, определяет полезность действия в состоянии, а не полезность самого состояния. Q-функция задается:

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}[G|S_0 = s, A_0 = a] = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s, A_0 = a \right].$$

В алгоритмах машинного обучения с подкреплением удобно представлять Q-функции в виде таблицы, чтобы сохранять оценки данных состояний и действий. Допустим, имеются два разных действия в двух разных состояниях. На основании ценности действий в конкретном состоянии можно выбрать какие действия являются предпочтительными в разных состояниях по Q-функции.



## Уравнение Беллмана и оптимальность

Существует много разных функций ценности для разных политик. Оптимальной функцией ценности  $V^*(s)$  называется функция ценности, которая обеспечивает максимальную ценность по сравнению со всеми другими функциями ценности. Аналогичным образом оптимальной политикой называется политика, обеспечивающая оптимальную функцию ценности. Оптимальную функцию ценности можно также представить:  $V^*(s) = \max_{\pi} V_{\pi}(s)$ .

Поскольку оптимальная функция ценности  $V^*(s)$  имеет более высокую ценность по сравнению со всеми остальными функциями ценности (то есть обеспечивает максимальный возврат), ее значение вычисляется как максимум по всем Q-функциям:  $V^*(s) = \max_a Q(s, a)$ .

При реализации метода Q-обучения используется Q-функция представленная в виде уравнения Беллмана. Представление Q-функции в виде уравнения Беллмана имеет важное значение в обучении с подкреплением. Уравнение Беллмана для Q-функции является рекуррентным соотношением, которое выражает связь между текущим значением Q-функции и её будущими значениями. Рекуррентное соотношение обеспечивает структуру, которая позволяет эффективно вычислять и обновлять значения Q-функции в процессе обучения. Запись Q-функции в виде уравнения Беллмана также помогает разделить задачу, что позволяет агенту лучше понимать, как его текущие действия влияют на будущую ценность состояния.

Q-функцию можно записать в виде уравнения Беллмана:  $Q_t(s, a) = R_t(s, a) + \max_{a'} Q_{t+1}(s', a')$ . Данное уравнение показывает, что максимальная будущая награда это награда  $R$  полученная за пребывания в данном состоянии плюс максимальная будущая награда для будущего состояния  $s'$  по всем действиям  $a'$ . Формально в терминах вероятностей переходов, политики  $\pi$  и мгновенных наград, уравнение Беллмана для Q-функции можно записать:

$$Q_{\pi}(s, a) = \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma \sum_{a'} Q_{\pi}(s', a') \right].$$

$P_{ss'}^a$  — вероятность перехода из состояния  $s$  в состояние  $s'$  при выполнении действия  $a$ .  $Q_{\pi}(s', a')$  представляет собой ожидаемую будущую награду, которую агент ожидает получить, начиная со состояния  $s'$ , выполняя действие  $a'$  и действуя далее согласно политики

π.

## Реализация Q-обучения

При реализации Q-обучения важно учитывать дилемму исследования и эксплуатации. Дилемма заключается в том, что агент должен находить баланс между исследованием и эксплуатацией, чтобы достичь оптимальных результатов. Исследование является процессом поиска новых стратегий, действий или решений с целью улучшения текущих оценок. Эксплуатация с другой стороны это использования текущих знаний и опыта для принятия решений. Если агент слишком сильно склоняется к исследованию, он может упустить возможность получить большие вознаграждения, используя уже известные стратегии. С другой стороны, слишком сильная ориентация на эксплуатацию может привести к упущению новых, возможно лучших стратегий.

Q-значение обновляется по следующей формуле:

$$Q(s, a) = Q(s, a) + (1 - \alpha)Q(s, a) + \alpha \left( R(s, a) + \gamma \max_{a'} Q(s', a') \right).$$

$\alpha$  — коэффициент доверия, который определяет, насколько агент будет "верить" новой информации при обновлении оценок Q. Если  $\alpha$  близко к 1, то агент сильно доверяет новым данным и быстро адаптируется к изменениям в среде. Если  $\alpha$  близко к 0, агент медленно обучается и менее подвержен влиянию новой информации. Выбор действия из состояния происходит с эпсилон жадной стратегией. Данная стратегия выбирает оптимальное действие среди уже исследованных. Оптимальным считается действие, обладающее наивысшей ценностью. Действие  $a$  в момент времени  $t$  с эпсилон жадной стратегией задастся:

$$a(t) = \begin{cases} \max_a Q_t(s, a), & \text{с вероятностью } 1 - \varepsilon \\ \text{любой } a(t) & \text{с вероятностью } \varepsilon \end{cases}$$

## Биржевой трейдер

Задача заключалась в создании программы для торговли акциями, способную изучать переменные рынка, генерировать действия (покупка, продажа, ожидание) и оценивать свою производительность. Данный трейдер был обучен методом Q-обучения на акциях компании Apple. Цель трейдера являлась максимизировать прибыль.

Для создания биржевого трейдера нужно было предобработать исторические данные акций компании Apple с помощью пакета 'pandas datareader' из yahoo!finance. Нужно было создать биржевого трейдера способного принимать решения о том, какое действие предпринять в текущей ситуации на фондовом рынке. Также нужно было создать метрику для оценки производительности трейдера. Оценка прибыльности трейдера задается отношением прибыли к инвестированному капиталу.

$$\text{Прибыльность} = \frac{\text{Общий капитал в конце торгов} - \text{инвестированный капитал}}{\text{инвестированный капитал}}$$

Для оценки эффективности трейдера сравнение прибыльности осуществляться с базовой прибыльностью, которая вычисляется, как разность покупки в первый день и продаже акций в последний день заданного набора данных.

$\epsilon$  выбирается равному 0.3, это означает, что в 30% случаях трейдер будет исследовать выбирая случайное действие. Если действие — покупка, вознаграждение будет нулевым, если действие — удержание, но акций в портфеле нет, вознаграждение также нулевое, а если действие — удержание и есть акции в портфеле, вознаграждение будет равняться разницей текущей цены и предыдущей цены. В случае если действие — продажа, но акций в портфеле нет, то вознаграждение будет равняться  $-100$ , а если действие — продажа и есть акции в портфеле, вознаграждение будет разницей текущей цены и цены первой покупки. Состояние агента это скорректированная цена закрытия.

## Результаты

Сравнение оценки базовой прибыльности и прибыльности производились на тренировочных и тестовых данных. При одном тренировочном цикле, обновляющий значения Q-таблицы и параметрами указанными выше на тренировочных данных прибыль составила 118.93%. Базовая прибыльность на тренировочных данных 106.83%, следовательно биржевой трейдер зарабатывает на 12% больше. На тестовых данных базовая прибыльность составила 23%, против 9% прибыльности данного биржевого трейдера на тестовых данных. Следовательно, трейдер с данными параметрами является слабым.

Для улучшения модели в состояние, которое определялась исключительно скорректированной ценой закрытия, добавляем SMA и линии Боллинджера. SMA — «простое скользящее среднее» (Simple Moving Average) — это технический индикатор, который используется для сглаживания ценовых данных и выявления общего тренда. Основная идея линий Боллинджера заключается в том, насколько цены актива разбросаны относительно своего среднего значения. Средняя линия Боллинджера это и есть SMA которое представляет собой среднюю цену актива за определённый период времени. Верхняя полоса отображает два стандартных отклонения от средней линии вверх, и соответственно нижняя полоса отображает два стандартных отклонения вниз. Данные технические индикаторы помогают трейдеру определить периоды высокой или низкой волатильности и принимать решения о входе или выходе из позиций. Например, пересечение цены актива с верхней полосой Боллинджера может сигнализировать о возможной перепроданности, а пересечение с нижней полосой — о возможной перекупленности.

При добавлении данных инструментов в состояние, вместе с увеличением количества циклов обновляющий Q-таблицу с 1 до 4, прибыльность становится значительно лучше, на тренировочных, и на тестовых данных. На тренировочных прибыльность составляет 755.28%, что на 648.49% лучше базовой прибыльности. На тестовых данных прибыльность 56.11%, что на 33% лучше базовой прибыльности. Следовательно можно сделать вывод, что настоящий биржевой трейдер с данными улучшениями работает лучше и может быть полезен, так как зарабатывает значительно больше, чем разность покупки в первый день и продажи акций в последний день заданного набора данных.

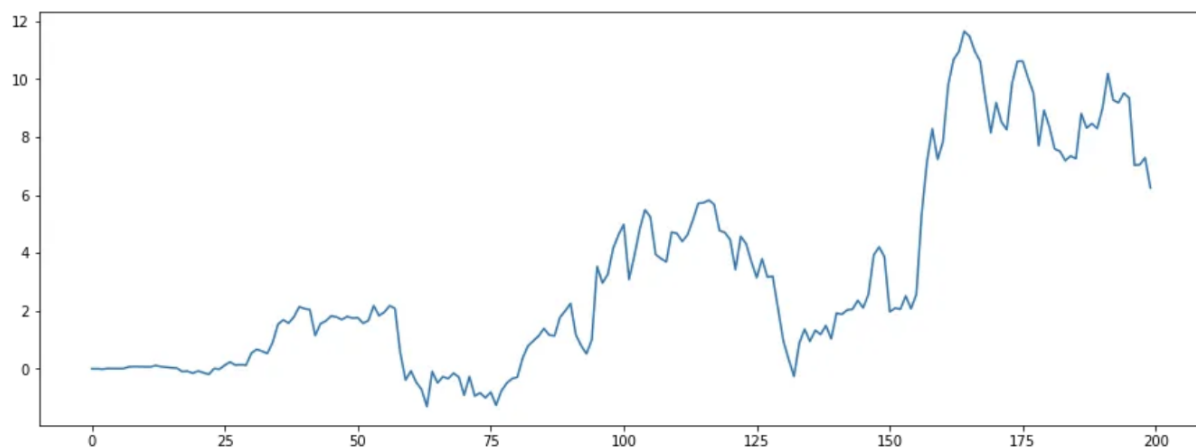


Рис. 1. График дохода

График Рис. 1 показывает доходность с момента входа в зависимости от времени.

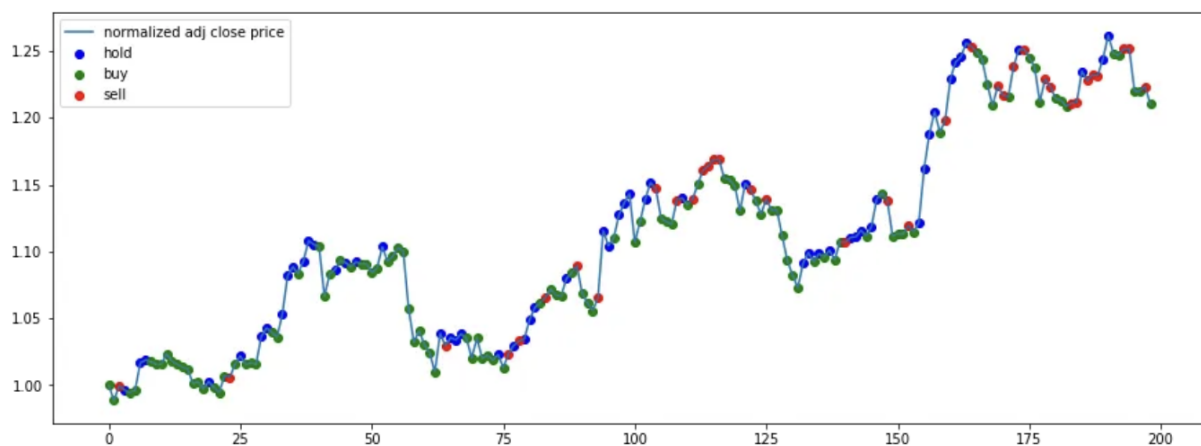


Рис. 2. График действий агента

График Рис. 2 отображает нормализованную цену закрытия с учетом корректировок, а также действия, выполненные обученным трейдером. Синие точки обозначают удержание, зеленые — покупку, а красные — продажу. Трейдер способен распознавать момент покупки на дне, продажу на пике и удержание, когда цена идет вверх.

## Заключение

В настоящей работе были изучены основы машинного обучения с подкреплением, а также метод стохастической оптимизации Q-обучение. При изучении метода Q-обучения был создан биржевой трейдер, который может покупать, удерживать и продавать акции на фондовом рынке принимая решения основываясь на финансовых технических индикаторах. Цель биржевого трейдера была максимизировать прибыль, что получилось сделать продемонстрировав это на акциях компании Apple.

Настоящая работа является началом темы по изучению применения стохастических методов в машинном обучении для решения финансовых задач. В следующем семестре планируется изучить другие методы такие как методы Монте Карло, методы временных различий и SARSA. Будет также возможность их сравнить и изучить более подробно, так как в данном семестре были изучены основы машинного обучения с подкреплением. В следующем семестре также планируется улучшение данной модели биржевого трейдера и проведение различных тестов на акциях других компаний и проверить оптимальность различных параметров, например количество тренировочных циклов.

При выполнении работы были также изучены материалы и примеры представленные в книге «Глубокое обучение с подкреплением на Python»[3] и "Recent Advances in Reinforcement Learning in Finance"[1]. При реализации кода, были также изучены различные новые функции и библиотеки в Python. Полный код и данные используемые в данной задаче доступны в репозитории GitHub[2].

Считаю, что цель курсовой работы по изучению основ машинного обучения с подкреплением и метода стохастической оптимизации выполнена. До этой работы, я ничего не знал об машинном обучении с подкреплением. Задача была изучена теоретически, и практически при реализации программы биржевого трейдера.

## Список литературы

1. Hambly B. Xu R. Yang H. Recent Advances in Reinforcement Learning in Finance. — L., 2023. — 65 p.
2. Милюшков Г. Г. GitHub: Georgmil/NIR. — <https://github.com/Georgmil/NIR7>. — 2023.
3. Равичандиран С. Глубокое обучение с подкреплением на Python. OpenAI Gym и TensorFlow для профи. — СПб. : Питер, 2019. — 251 с.

## Приложение

```

1 import time
2 import datetime
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 import data_process as d
7 import pandas_datareader.data as web
8
9 get_ipython().run_line_magic('matplotlib', 'inline')
10 def initialize_q_mat(all_states, all_actions):
11     #Creating Q-table
12     states_size = len(all_states)
13     actions_size = len(all_actions)
14
15     q_mat = np.random.rand(states_size, actions_size)/1e9
16     q_mat = pd.DataFrame(q_mat, columns=all_actions.keys())
17
18     q_mat['states'] = all_states
19     q_mat.set_index('states', inplace=True)
20
21     return q_mat
22
23 def act(state, q_mat, threshold=0.2, actions_size=3):
24     #Action function
25     if np.random.uniform(0,1) < threshold: # go random

```

```

26         action = np.random.randint(low=0, high=actions_size)
27     else:
28         action = np.argmax(q_mat.loc[state].values)
29     return action
30
31 def get_return_since_entry(bought_history, current_adj_close):
32     #Return
33     return_since_entry = 0.
34
35     for b in bought_history:
36         return_since_entry += (current_adj_close - b)
37     return return_since_entry
38
39 def visualize_results(actions_history, returns_since_entry):
40     #Visualize results
41     f, (ax1, ax2) = plt.subplots(2, 1, figsize=(15,12))
42
43     ax1.plot(returns_since_entry)
44
45     days, prices, actions = [], [], []
46     for d, p, a in actions_history:
47         days.append(d)
48         prices.append(p)
49         actions.append(a)
50
51     ax2.plot(days, prices, label='normalized adj close price')
52     hold_d, hold_p, buy_d, buy_p, sell_d, sell_p = [], [], [], [], [], []
53     for d, p, a in actions_history:
54         if a == 0:
55             hold_d.append(d)
56             hold_p.append(p)
57         if a == 1:
58             buy_d.append(d)
59             buy_p.append(p)
60         if a == 2:
61             sell_d.append(d)
62             sell_p.append(p)
63
64     ax2.scatter(hold_d, hold_p, color='blue', label='hold')

```



```

65     ax2.scatter(buy_d, buy_p, color='green', label='buy')
66     ax2.scatter(sell_d, sell_p, color='red', label='sell')
67     ax2.legend()
68
69 def get_invested_capital(actions_history, returns_since_entry):
70     #Get capital
71     invest = []
72     total = 0
73     return_invest_ratio = None
74     for i in range(len(actions_history)):
75         a = actions_history[i][2]
76         p = actions_history[i][1]
77
78         try:
79             next_a = actions_history[i+1][2]
80         except:
81             #print('end')
82             break
83         if a == 1:
84             total += p
85             #print(total)
86             if next_a != 1 or (i==len(actions_history)-2 and next_a==1):
87                 invest.append(total)
88                 total = 0
89     if invest:
90         return_invest_ratio = returns_since_entry[-1]/max(invest)
91         print('invested capital {}, return/invest ratio {}'.format(max(
92             invest), return_invest_ratio))
93     else:
94         print('no buy transactions, invalid training')
95     return return_invest_ratio
96
97 def get_base_return(data):
98     #Benchmark return(Bazovaja pribilnost)
99
100     start_price, _ = data[0]
101     end_price, _ = data[-1]
102     return (end_price - start_price)/start_price

```

```

103 start = datetime.datetime(2014, 1, 1)
104 end = datetime.datetime(2018, 1, 1)
105 train_df, test_df = d.get_stock_data('AAPL', start, end, 0.8)
106
107
108 train_df.head()
109
110
111 test_df = d.create_df(test_df, 3)
112 test_df = d.create_state_df(test_df, price_states_value, bb_states_value,
113                             close_sma_ratio_states_value)
114
115
116 all_actions = {0: 'hold', 1: 'buy', 2: 'sell'}
117
118
119 train_df = d.create_df(train_df, 3)
120 price_states_value, bb_states_value, close_sma_ratio_states_value = d.
121     get_states(train_df)
122
123 train_df = d.create_state_df(train_df, price_states_value, bb_states_value,
124                             close_sma_ratio_states_value)
125
126
127 all_states = d.get_all_states(price_states_value, bb_states_value,
128                             close_sma_ratio_states_value)
129
130 states_size = len(all_states)
131
132
133 def train_q_learning(train_data, q, alpha, gamma, episodes):
134     #training Q-table
135     actions_history = []
136     num_shares = 0
137     bought_history = []
138     returns_since_entry = [0]
139     for ii in range(episodes):
140         actions_history = []
141         num_shares = 0
142         bought_history = []
143         returns_since_entry = [0]
144         days=[0]
145         for i, val in enumerate(train_data):
146             current_adj_close, state = val
147             try:

```

```

138         next_adj_close, next_state = train_data[i+1]
139     except:
140         break
141
142     if len(bought_history) > 0:
143         returns_since_entry.append(get_return_since_entry(
144             bought_history, current_adj_close))
145     else:
146         returns_since_entry.append(returns_since_entry[-1])
147
148     # decide action
149     if alpha > 0.1:
150         alpha = alpha/(i+1)
151     action = act(state, q, threshold=alpha, actions_size=3)
152
153     # get reward
154     if action == 0: # hold
155         if num_shares > 0:
156             prev_adj_close, _ = train_data[i-1]
157             future = next_adj_close - current_adj_close
158             past = current_adj_close - prev_adj_close
159             reward = past
160         else:
161             reward = 0
162
163     if action == 1: # buy
164         reward = 0
165         num_shares += 1
166         bought_history.append((current_adj_close))
167
168     if action == 2: # sell
169         if num_shares > 0:
170             bought_price = bought_history[0]
171             reward = (current_adj_close - bought_price)
172             bought_history.pop(0)
173             num_shares -= 1
174         else:
175             reward = -100

```

```

176         actions_history.append((i, current_adj_close, action))
177
178         # update q table
179         q.loc[state, action] = (1.-alpha)*q.loc[state, action] + alpha*(
reward+gamma*(q.loc[next_state].max()))
180     print('End of Training!')
181     return q, actions_history, returns_since_entry
182
183
184 def eval_q_learning(test_data, q):
185     #reviewing Q-table
186     actions_history = []
187     num_shares = 0
188     returns_since_entry = [0]
189     bought_history = []
190
191     for i, val in enumerate(test_data):
192         current_adj_close, state = val
193         try:
194             next_adj_close, next_state = test_data[i+1]
195         except:
196             print('End of data! Done!')
197             break
198
199         if len(bought_history) > 0:
200             returns_since_entry.append(get_return_since_entry(bought_history
, current_adj_close))
201         else:
202             returns_since_entry.append(returns_since_entry[-1])
203
204         # decide action
205         action = act(state, q, threshold=0, actions_size=3)
206
207         if action == 1: # buy
208             num_shares += 1
209             bought_history.append((current_adj_close))
210         if action == 2: # sell
211             if num_shares > 0:
212                 bought_price = bought_history[0]

```

```

213         bought_history.pop(0)
214         num_shares -= 1
215
216         actions_history.append((i, current_adj_close, action))
217
218     return actions_history, returns_since_entry
219
220
221 train_df.head()
222
223
224
225 np.random.seed(12)
226 q_init = initialize_q_mat(train_df['norm_adj_close_state'].unique(),
227                             all_actions)
228 print('Initializing q')
229 print(q_init)
230
231
232 train_data = np.array(train_df[['norm_adj_close', 'norm_adj_close_state']])
233 q, train_actions_history, train_returns_since_entry = train_q_learning(
234     train_data, q_init, alpha=0.8, gamma=0.95, episodes=1)
235
236 visualize_results(train_actions_history, train_returns_since_entry)
237 get_invested_capital(train_actions_history, train_returns_since_entry)
238 print('base return/invest ratio {}'.format(get_base_return(train_data)))
239
240
241 # ## test data
242
243 test_data = np.array(test_df[['norm_adj_close', 'norm_adj_close_state']])
244 test_actions_history, test_returns_since_entry = eval_q_learning(test_data,
245     q)
246
247 visualize_results(test_actions_history, test_returns_since_entry)
248 get_invested_capital(test_actions_history, test_returns_since_entry)
249 print('base return/invest ratio {}'.format(get_base_return(test_data)))
250

```

```

249
250 # # improving model
251
252 np.random.seed(12)
253 q = initialize_q_mat(all_states, all_actions)/1e9
254 print('Initializing q')
255 print(q[:3])
256
257 train_data = np.array(train_df[['norm_adj_close', 'state']])
258 q, train_actions_history, train_returns_since_entry = train_q_learning(
    train_data, q, alpha=0.8, gamma=0.95, episodes=1)
259
260 visualize_results(train_actions_history, train_returns_since_entry)
261 get_invested_capital(train_actions_history, train_returns_since_entry)
262 print('base return/invest ratio {}'.format(get_base_return(train_data)))
263
264 test_data = np.array(test_df[['norm_adj_close', 'state']])
265 test_actions_history, test_returns_since_entry = eval_q_learning(test_data,
    q)
266
267 visualize_results(test_actions_history, test_returns_since_entry)
268 get_invested_capital(test_actions_history, test_returns_since_entry)
269 # print('invested capital {}, return/invest ratio {}'.format(
    invested_capital, return_invest_ratio))
270 print('base return/invest ratio {}'.format(get_base_return(test_data)))

```

### Вспомогательный код с функциями:

```

1 import datetime
2 import numpy as np
3 import pandas as pd
4 import pandas_datareader.data as web
5
6 def get_stock_data(symbol, start, end, train_size=0.8):
7
8     #Get stock data in the given date range
9     df = web.DataReader(symbol, 'yahoo', start, end)
10
11     train_len = int(df.shape[0] * train_size)

```

```

12
13     if train_len > 0:
14         train_df = df.iloc[:train_len, :]
15         test_df = df.iloc[train_len:, :]
16         return train_df, test_df
17     else:
18         return df
19
20
21 def get_bollinger_bands(values, window):
22     #Return upper and lower Bollinger Bands.
23
24     rm = values.rolling(window=window).mean()
25     rstd = values.rolling(window=window).std()
26
27     band_width = 2. * rstd / rm
28     return band_width.apply(lambda x: round(x,5))
29
30 def get_adj_close_sma_ratio(values, window):
31
32     #Return the ratio of adjusted closing value to the simple moving average
33     .
34
35     rm = values.rolling(window=window).mean()
36     ratio = values/rm
37     return ratio.apply(lambda x: round(x,5))
38
39 def discretize(values, num_states=10):
40     #Convert continuous values to integer state
41
42     states_value = dict()
43     step_size = 1./num_states
44     for i in range(num_states):
45         if i == num_states - 1:
46             states_value[i] = values.max()
47         else:
48             states_value[i] = values.quantile((i+1)*step_size)
49     return states_value

```

```

50 def value_to_state(value, states_value):
51     #Convert values to state
52
53     if np.isnan(value):
54         return np.nan
55     else:
56         for state, v in states_value.items():
57             if value <= v:
58                 return str(state)
59         return 'value out of range'
60
61 def create_df(df, window=3):
62
63     #Create a dataframe w
64
65     # get bollinger value
66     bb_width = get_bollinger_bands(df['Adj Close'], window)
67     # get the ratio of close price to simple moving average
68     close_sma_ratio = get_adj_close_sma_ratio(df['Close'], window)
69
70     df['bb_width'] = bb_width
71     df['close_sma_ratio'] = close_sma_ratio
72
73     # drop missing values
74     df.dropna(inplace=True)
75
76     # normalize close price
77     df['norm_adj_close'] = df['Adj Close']/df.iloc[0,:]['Adj Close']
78     df['norm_bb_width'] = df['bb_width']/df.iloc[0,:]['bb_width']
79     df['norm_close_sma_ratio'] = df['close_sma_ratio']/df.iloc[0,:]['
close_sma_ratio']
80
81     return df
82
83 def get_states(df):
84
85     # discretize values
86     price_states_value = discretize(df['norm_adj_close'])
87     bb_states_value = discretize(df['norm_bb_width'])

```



```

88     close_sma_ratio_states_value = discretize(df['norm_close_sma_ratio'])
89
90     return price_states_value, bb_states_value, close_sma_ratio_states_value
91
92
93 def create_state_df(df, price_states_value, bb_states_value,
94     close_sma_ratio_states_value):
95
96     df['bb_width_state'] = df['bb_width'].apply(lambda x : value_to_state(x,
97         bb_states_value))
98     df['close_sma_ratio_state'] = df['close_sma_ratio'].apply(lambda x :
99         value_to_state(x, close_sma_ratio_states_value))
100     df['norm_adj_close_state'] = df['norm_adj_close'].apply(lambda x :
101         value_to_state(x, price_states_value))
102
103     df['state'] = df['norm_adj_close_state'] + df['close_sma_ratio_state'] +
104         df['bb_width_state']
105     df.dropna(inplace=True)
106     return df
107
108
109 def get_all_states(price_states_value, bb_states_value,
110     close_sma_ratio_states_value):
111
112     states = []
113     for p, _ in price_states_value.items():
114         for c, _ in close_sma_ratio_states_value.items():
115             for b, _ in bb_states_value.items():
116                 state = str(p) + str(c) + str(b)
117                 states.append(str(state))
118
119     return states

```