ТЕХНОЛОГИЧЕСКАЯ КАРТА ЗАНЯТИЯ

Тема занятия: Композиции алгоритмов. Бэггинг, случайный лес, стекинг и бустинг.

Аннотация к занятию: В первой части занятия обучающиеся знакомятся с идеей построения композиции и её теоретическим обоснованием. Узнают о бэггинге, случайном лесе, стекинге и бустинге — конкретных алгоритмах на основе идеи композиции. Во второй части занятия создают интеллект-карту или схему с ответом на предложенные вопросы.

Цель занятия: Изучение различных композиций алгоритмов (идеи из принципа Кондорсе и эксперимента Гальтона, бэггинг, случайный лес, стекинг, бустинг).

Задачи занятия:

- познакомить обучающихся с идеями различных деятелей о композиции алгоритмов, способствовать пониманию информации на простых примерах;
- показать разнообразие композиций и их основные преимущества и недостатки;
- научить использовать бэггинг, случайный лес, стекинг и бустинг в решении задач машинного обучения.

Ход занятия

Этап занятия	Время	Деятельность педагога	Комментарии, рекомендации для педагогов
Организационный этап	1 мин.	Здравствуйте! Сегодня нас ждёт знакомство с различными идеями построения композиции алгоритмов. Все готовы к занятию?	Обеспечение полной готовности аудитории к работе на занятии
Постановка цели и задач занятия. Мотивация учебной деятельности обучающихся	7 мин.	Мы уже изучили несколько моделей машинного обучения. **Вопрос** Какие модели машинного обучения вы помните? **Возможный ответ обучающихся** Метод ближайших соседей, линейные алгоритмы и решающее дерево. У каждого из этих алгоритмов есть свои преимущества и недостатки. На самом деле, в нашем распоряжении есть целый «зоопарк» моделей, и все они уникальны. Именно поэтому теоретики и практики используют несколько алгоритмов одновременно. **Вопрос** Как вы думаете, для чего это нужно?	Создание мотивации учебной деятельности. Подведение к теме занятия

		Возможный ответ обучающихся Чтобы вместе они создавали сильную модель без недостатков. Раз каждый алгоритм по-своему анализирует данные, то несколько моделей могут получить более общую картину. Так мы подходим к идее композиции алгоритмов. Она очень простая. Каждый конкретный алгоритм, даже самый качественный, всё равно иногда ошибается при решении задачи. Но если мы построим 10 алгоритмов, и спросим ответ у каждого из них, то сможем принять взвешенное решение с учётом мнения каждого алгоритма. Получается такая демократия.	
Изучение нового материала	40 мин.	Теория для построения композиции алгоритмов разрабатывалась самыми разными людьми. Вот, например, Николя де Кондорсе — видный французский деятель эпохи Просвещения, который анализировал демократические процедуры (например, выборы и другие голосования). Кондорсе был первым, кто попытался применить математику к социологии и общественным наукам в целом. Помимо прочего, Кондорсе вырабатывал принципы работы суда присяжных. Представьте, что коллегия присяжных состоит из N человек. Тогда, если каждый из присяжных по отдельности голосует за верный вердикт с вероятностью больше чем ½, то вероятность того, что большинство из них оказывается право, стремится к единице с ростом количества присяжных. В терминах машинного обучения, если каждый алгоритм угадывает правильный ответ на конкретном объекте более чем в 50% случаев и количество	Активные действия учащихся с объемом изучения. Вызов познавательного интереса к предмету. Для справки: Xgboost — это библиотека с открытым исходным кодом, используемая в машинном обучении и предоставляющая функциональность для решения задач,

		алгоритмов стремится к бесконечности, то с вероятностью, стремящейся к 1, большинство из алгоритмов окажется право. Забавная иллюстрация теоремы Кондорсе — эксперимент, который в конце XIX века провёл английский учёный Фрэнсис Гальтон. Кстати, именно он ввёл термин «регрессия». Эксперимент Гальтона состоял в следующем. Как-то раз он привёл с собой на рынок быка весом 1198 фунтов и попросил прохожих угадать, сколько весит бык. Он опросил примерно 800 человек, и никто из них не угадал точный вес быка. Зато когда Гальтон посчитал среднее всех ответов, оно оказалось равным 1197 фунтов, то есть отличалось от правильного ответа всего на 1 фунт. Представим, что у нас вместо быка — задача машинного обучения, а вместо крестьян — несколько алгоритмов. Метод решения, при котором усредняются результаты предсказания, в машинном обучении называется методом простого голосования. Пусть a_1, a_2, ..., a_n — несколько различных обученных алгоритмов, скажем, на задаче классификации. Возьмём какой-то объект x. Прогоним этот объект через все наши алгоритмы. Тогда композиция алгоритмов a_1, a_2, ..., a_n отнесёт объект x к тому классу, за который проголосовало большинство из a_1, a_2, ..., a_n — в случае задачи классификации. Если перед нами изначально не задача классификации, а задача регрессии, то можно просто	связанных с регуляризацией градиентного бустинга. Библиотека поддерживается языками программирования C++, Java, Python, R, Julia, Perl и Scala. lightGBM — сокращение от Light Gradient Boosting Machine, представляет собой бесплатную распределённую структуру повышения градиента с открытым исходным кодом для машинного обучения, первоначально разработанную Microsoft. CatBoost — это программная библиотека с открытым исходным кодом, разработанная Яндексом. Он предоставляет структуру повышения

		усреднить ответы всех алгоритмов, как Гальтон усреднял ответы крестьян. Вооружившись этой идеей, подумаем, как лучше всего построить много различных алгоритмов, каждый из которых будет решать одну и ту же задачу. Возникает проблема. Все построенные алгоритмы одного типа на одной и той же выборке получатся одинаковыми. Решение в том, чтобы выбрать один класс алгоритмов — например, решающее дерево с фиксированной глубиной — и каждый новый алгоритм строить на новой выборке, которая не очень сильно отличается от старой. Когда мы рассматривали решающие деревья, то отмечали их склонность к переобучению. Так вот, если мы внесём в выборку разнообразие для каждого алгоритма, то на этих разнообразных выборках построятся разные решающие деревья, которые, тем не менее, будут всё ещё решать одну и ту же исходную задачу, пусть и с небольшими изменениями. Осталось понять, как, имея одну обучающую выборку, сформировать большое количество новых выборок, которые похожи на изначальную. Здесь нам поможет генерация новых выборок — бутстреп. Пусть у нас есть 12 объектов — это разноцветные шарики слева. Давайте по очереди выбирать шарики из этой выборки и класть их в новую обучающую выборку. Мы не будем запрещать выбор одного и того же шарика из изначальной выборки несколько раз: как видите, в получившейся новой выборке один и тот же цвет может встречаться два или более раз. Так мы построим одну выборку.	градиента, которая, среди прочего, пытается решать категориальные функции, используя альтернативу, основанную на перестановках, по сравнению с классическим алгоритмом.

		Затем построим ещё несколько выборок. Как видите, все выборки немного разные, но напоминают изначальную выборку. Получившиеся выборки называются бутстрепными. На картинке показано, что в каждой бутстрепной выборке по 8 элементов, но на самом деле обычно в бутстрепную выборку берётся столько элементов, сколько и в изначальной выборке. На последнем шаге мы строим отдельный алгоритм для каждой выборки, например, решающее дерево. Как строить один-единственный алгоритм, мы уже знаем. Итоговое решение о классификации объекта принимается голосованием всех построенных алгоритмов. Такой способ композиции алгоритмов называется бэггинг (от английского bag — сумка). Для начала нам необходимо зафиксировать базовый алгоритм — пусть это будет решающее дерево. Зафиксируем число деревьев, скажем, 100. Для обучения каждого дерева нам нужна обучающая выборка. В качестве неё мы берём бутстрепную выборку из исходной обучающей выборки, которая будет похожа на исходную. Итоговое решение принимается большинством голосов. Из-за различий в обучающих подвыборках каждого из ста деревьев, все алгоритмы получаются немного разными. Напомним, что сильная сторона решающего дерева — это вариативность, а слабая — нестабильность и склонность к переобучению. Но в нашем алгоритме нестабильность больше не является недостатком, так как решение принимается на основе сотни различных предсказаний. Наоборот, недостатком была бы излишняя бедность	

		базовых алгоритмов. Поэтому для бэггинга в качестве базовых алгоритмов лучше выбирать деревья с максимальной глубиной. Описанный алгоритм называют случайным лесом, если в качестве базовых алгоритмов выступают решающие деревья. Случайный лес хорошо подходит для решения конкурсных задач. Стоит отметить важное отличие случайного леса от простого бэггинга над решающими деревьями. Как и в обычном бэггинге, для каждого дерева мы генерируем свою бутстрепную обучающую выборку. При построении каждого дерева мы вносим дополнительный «случайный» элемент: при нахождении лучшего решающего правила для сплита мы в каждой вершине выбираем правило не из всех возможных признаков, а из случайного подмножества признаков небольшого размера. Это ещё сильнее отличает решающие деревья друг от друга, а разнообразие, в свою очередь, повышает качество итогового алгоритма. Основной недостаток всех композиций и решающего леса в частности — громоздкость: чтобы вычислить ответ, необходимо опросить все базовые алгоритмы, а затем усреднить предсказание. Поэтому в производстве, если это возможно, стараются обойтись без огромных композиций, особенно когда важна скорость принятия решений. Тем не менее, если нужно выжать из задачи максимальное качество, то решающий лес обязательно нужно попробовать. Кстати, вычисления для решающего леса можно выполнять параллельно, потому что построение базовых алгоритмов друг от друга никак не зависит.	

		Стекинг — очень полезная техника построения композиции из небольшого количества сильных алгоритмов. Она повышает качество предсказаний. Когда вы будете участвовать в конкурсах по машинному обучению, стекинг наверняка поможет набрать десятые доли процента качества, которые часто определяют разницу между первым и десятым местом в соревновании. Представьте, что вы решаете задачу предсказания стоимости квартиры. У вас есть обучающая выборка, на основе которой вы построили три алгоритма: a, b и c. Например, линейную регрессию, метод ближайших соседей и случайный лес. Все они дают какое-то качество на тестовой выборке: один больше, другой меньше. Среди них нет алгоритма, который бы всегда работал лучше остальных: какие-то задачи лучше решает первый алгоритм, другие — второй или третий. Для нас это не новость: когда мы строили случайный лес, разные алгоритмы тоже работали лучше в разных участках признакового пространства. Мы же хотим создать супералгоритм из алгоритмов a, b и c, который будет лучше их троих. Когда мы строили бэггинг, то брали и усредняли все три алгоритма. Это работало, потому что алгоритмы компенсировали ошибки друг друга. Но у этого подхода есть важный недостаток. Если алгоритм a в среднем работает лучше, чем алгоритмы b и c, то их среднее арифметическое будет иметь качество лучше, чем b и c, но, скорее всего, хуже, чем a, потому что в большинстве случаев b и c будут только портить картину. Мы хотим понимать, какому из алгоритмов больше верить.	

		Возможно, вы уже догадались, в чём смысл стекинга. Если у нас есть три алгоритма, то давайте предскажем значения a(x), b(x), c(x) на элементах обучающей выборки, а затем предоставим новому метаалгоритму принимать окончательное решение, используя a(x), b(x) и c(x). Таким образом мы интерпретируем a(x), b(x) и c(x) как новые признаки нашего объекта x. Затем мы должны обучить на этих признаках метаалгоритм m(a,b,c), с помощью которого мы затем предскажем окончательный ответ. То есть мы «смешиваем» результаты трёх уже построенных алгоритмов с помощью нового, четвёртого алгоритма M. И надеемся, что метаалгоритм лучше нас поймёт, какому из алгоритмов придавать больший вес в зависимости от значений, которые они предсказывают. На практике мы сталкиваемся с трудностями. Возникает вопрос: на каких данных мы должны построить обучение? Пусть у нас есть простая задача классификации: всего 8 объектов, у каждого какие-то признаки. Есть целевая переменная Y со значениями 0 или 1, для предсказания которой мы построили три алгоритма: a, b и c. Для каждого объекта обучающей выборки мы выписали в таблицу вероятности того, что Y равно 1, предсказанные алгоритмами a, b и c. И вот, представьте, что вы — это тот самый метаалгоритм. На практике мы сталкиваемся с трудностями. Возникает вопрос: на каких данных мы должны построить обучение? Пусть у нас есть простая задача классификации: всего 8 объектов, у каждого какие-то признаки. Есть целевая переменная Y со значениями 0 или 1, для предсказания	

		которой мы построили три алгоритма: a, b и c. Для каждого объекта обучающей выборки мы выписали в таблицу вероятности того, что Y равно 1, предсказанные алгоритмами a, b и c. И вот, представьте, что вы — это тот самый метаалгоритм. Вы видите перед собой вот такую обучающую выборку. Столбцы a(x), b(x), c(x) — это матрица объекты-признаки, а столбец y — целевая переменная. Теперь посмотрите на матрицу объекты-признаки внимательно. Видно, что первый алгоритм лучше всех предсказывает правильные ответы на обучающей выборке — его вероятности ближе всего к правильному ответу. Поэтому метаалгоритму при обучении больше всего понравится алгоритм a(x). Алгоритмы b и c не хочется использовать вообще, раз у нас есть алгоритм a, который предсказывает результат точнее всего. Такой подход может привести к катастрофе. Если первый алгоритм точно предсказал ответы на обучающей выборке, это вовсе не значит, что на тестовой выборке он сработает хорошо. Первый алгоритм мог сильно переобучиться, а третий — почти не переобучиться. Хотя метаалгоритм и посчитает, что алгоритм a самый полезный, на самом деле самым полезным и точным может оказаться алгоритм c. Метаалгоритм отдаст предпочтение не самому качественному алгоритму, а самому переобученному, чего мы, естественно, не хотим. Как решить эту проблему? Очень просто. Проблема в том, что метаалгоритм обучался на тех же данных, что и базовые алгоритмы. В качестве альтернативы есть следующая схема.	

		Разделим наши обучающие данные на две части: data_1 и data_2. Сначала мы берём набор данных data_1 и обучаем на них базовые алгоритмы a,b и c. Затем для обучения метаалгоритма мы берём набор данных data_2, вычисляем на них признаки, соответствующие алгоритмам a,b и c и обучаем метаалгоритм M на этих данных. Последний этап — измерение итогового качества модели на отдельной тестовой выборке — без этого никуда. У схемы есть преимущества: она лишена проблем с переобучением и быстро обучается. Но есть и проблемы. Главная из них — необходимость очень большого количества данных для обучения. Если мы резервируем блок данных для обучения метаалгоритма, то базовым алгоритмам данных может просто не хватить. Тем не менее, эта схема работает. Её называют блендинг. Граница между понятиями стекинга и блендинга размытая: обычно под блендингом понимают простые схемы стекинга. Посмотрим на преимущества и недостатки стекинга. Предположим, мы создали несколько алгоритмов различного качества и хотим понять, какое качество будет у их композиции после стекинга. Оказывается, качество лишь незначительно превзойдёт качество лучшего из алгоритмов, особенно если базовые алгоритмы довольно сильные. На диаграмме вы видите результаты стекинга (по двум различным схемам) в	

		применении к 9 весьма сильным моделям. Первая схема стекинга — блендинг — дала тот же результат, что и наилучший алгоритм, то есть прироста нет. А стекинг с кросс-валидацией дал очень небольшой, но всё же прирост по сравнению с лучшим алгоритмом. Из недостатков можно выделить незначительный прирост (если он вообще есть) и большую громоздкость процедуры стекинга. В завершение я дам советы о том, как строить и использовать стекинг. Во-первых, в качестве базовых алгоритмов мы рекомендуем брать несколько разноплановых и довольно сложных моделей. Например, случайные леса с разной глубиной, линейные алгоритмы и модели градиентного бустинга, о которых пойдёт речь дальше. Вторая рекомендация: аккуратно подходить к выбору метаалгоритма. Попробуйте разные варианты: решающее дерево, случайный лес, линейные алгоритмы. Тщательно подбирайте параметры метаалгоритма, например, с помощью кросс-валидации. Третье. Используйте стекинг, если данных много. Если их недостаточно, он вряд ли хорошо сработает. Третий и самый мощный метод композиции алгоритмов — бустинг. Как и в бэггинге, мы хотим построить множество алгоритмов для решения исходной задачи и принимать	

		решение голосованием. Если в бэггинге мы строили алгоритмы независимо друг от друга, то в бустинге каждый следующий алгоритм компенсирует ошибки всех предыдущих. Одна из самых известных разновидностей бустинга — градиентный бустинг над решающими деревьями. «Градиентный» означает, что для подсчёта каждого следующего алгоритма в композиции мы используем градиент. «Над решающими деревьями» значит, что в качестве базовых алгоритмов мы используем решающие деревья. Более подробно о том, как работает градиентный бустинг, вы можете прочитать в заметках к лекции. На первый взгляд кажется, что такой алгоритм — это прямой путь к переобучению, ведь каждая следующая модель всё больше и больше подгоняет итоговый результат под обучающую выборку. На практике оказывается, что градиентный бустинг прекрасно проявляет себя в сложных задачах машинного обучения, если обучающая выборка достаточно большая. Поговорим об особенностях градиентного бустинга. Во-первых, базовые алгоритмы должны быть простыми и быстро обучаемыми, поскольку обучение бустинга — задача сложная и долгая. Во-вторых, в отличие от бэггинга, здесь нет требования на излишнюю сложность базовых алгоритмов. В бэггинге сложные алгоритмы нужны, потому что вся сложность заключена именно в самих базовых алгоритмах. В случае бустинга дополнительная сложность и вариативность достигается за счёт того, что каждый следующий алгоритм пытается исправить ошибку предыдущих. Например, если	

		речь идёт о решающих деревьях, то для бустинга достаточно деревьев с меньшей глубиной. Три самые популярные реализации градиентного бустинга с открытым кодом — XGBoost, LightGBM и CatBoost от Яндекса. Мы опробуем их на практике и сравним между собой. Напоследок поговорим о преимуществах и недостатках бустинга. Во-первых, бустинг точно восстанавливает закономерности в данных, лучше любого другого алгоритма машинного обучения. Это происходит благодаря высокой сложности получающихся моделей. Более того, эффект переобучения оказывается не таким сильным, как можно было предположить. Поэтому бустинг часто используют на конкурсах по машинному обучению. Ещё одно важное преимущество бустинга — универсальность. Он работает для любой задачи и любой функции потерь: классификации, регрессии и любой другой. Недостаток бустинга совпадает с главным недостатком решающего леса. Для вычисления ответа алгоритма необходимо вычислить ответ каждого базового алгоритма по-отдельности, а это порой занимает слишком много времени. Но в случае, если вычислительные ресурсы позволяют, бустинг становится очень мощным оружием в руках разработчика машинного обучения.	

Закрепление изученного материала	30 мин.	Творческая работа. Ответы на вопросы в формате интеллект-карты или картинок. **Вопросы:** 1. Что такое композиции алгоритмов? Какие композиции вы знаете? 2. Случайный лес основан на принципе бэггинга — усреднения предсказания нескольких независимых моделей. Но вот проблема: обучающая выборка у нас одна. Как мы можем получить независимые модели? Почему это не приводит к проблемам? 3. Какие гиперпараметры вы бы стали настраивать для борьбы с переобучением у градиентного бустинга? **Предполагаемые ответы:** 1. Классические модели машинного обучения несовершенны. Например, с помощью одного дерева решений вряд ли можно решить сложную задачу. Но если собрать много базовых моделей, то из них может родиться что-то гораздо более интересное. Есть несколько стратегий построения ансамблей. Бэггинг. Его идея в том, чтобы просто агрегировать предсказания, выдаваемые базовыми моделями (например, усреднять или брать самое популярное предсказание). Чтобы результат получился качественным, нужно, чтобы каждая модель была достаточно сильной (основной пример бэггинга — случайный лес — использует глубокие, переобученные решающие деревья). Бустинг. Эта стратегия предполагает, что базовые модели строятся последовательно, и каждая следующая уточняет	Выполнение практического задания. Рекомендации учителю: Обучающихся рекомендуется разделить на группы. Каждая группа должна создать свою интеллект-карту или схему (картинку). Для справки: Диаграмма связей, известная также как интеллект-карта, карта мыслей или ассоциативная карта — метод структуризации концепций с использованием графической записи в виде диаграммы. Как создавать интеллект-карту

		предсказание всех предыдущих. Обычно в качестве базовых берутся слабые модели, чаще всего деревья небольшой глубины. Стекинг. Предлагается использовать предсказания базовых моделей в качестве новых признаков, а поверх них обучить ещё какую-нибудь модель. Стекать можно что угодно, хоть нейросети, хоть детерминированные пайплайны извлечения фичей. Стекинг вполне встречается и в продакшене, хотя часто от него стараются избавляться в пользу end-to-end обучения, то есть одной большой модели, которая обучается целиком, а не поэтапно. 2. На практике оказывается, что строгое выполнение предположения о независимости необязательно. Достаточно, чтобы алгоритмы были в некоторой степени не похожи друг на друга. Достаточная непохожесть обеспечивается тем, что при обучении каждого дерева мы берём: ● случайное подмножество обучающей выборки; ● случайное подмножество признаков. 3. Объяснение обычно строится на основе разложения ошибки на смещение и разброс. Грубо говоря, смещение показывает, насколько модели, обученные на разных выборках, в среднем ошибаются, а разброс — насколько разными будут модели, обученные на разных выборках. Каждая следующая базовая модель в бустинге обучается так, чтобы уменьшить общую ошибку всех своих предшественников. Как следствие, итоговая композиция будет иметь меньшее смещение, чем каждый отдельный	

		базовый алгоритм (уменьшение разброса также может происходить, но не обязательно). Переобучение обычно характеризуется низким смещением и высоким разбросом. Поскольку просто увеличением числа базовых моделей мы разброс не уменьшим, нужно будет понизить разброс каждой из них. В конечном итоге это сводится к тому, что модели надо сделать проще. Если речь идёт о бустинге над деревьями, то стоит уменьшить их глубину (кстати, по этой причине в бустинге обычно используются деревья небольшой глубины). Так как базовые модели становятся проще, имеет смысл увеличить их количество, но при этом нам не хочется, чтобы они слишком быстро приблизили таргет на обучающей выборке (это тоже было бы переобучением), поэтому бывает полезно одновременно уменьшить learning rate (темп обучения).	
Этап подведения итогов занятия (рефлексия)	10 мин.	**Подведём итоги.** **Вопросы для обсуждения** ● Что вдохновило исследователей машинного обучения на эту идею: принцип Кондорсе и эксперимент Гальтона? ● Как работает алгоритм случайный лес? ● Для чего используется бустинг? ● В чём преимущества стекинга? **Вопросы для рефлексии** ● Какая композиция вызвала у вас наибольший интерес? ● Как вы оцениваете свою работу в микрогруппах?	Способствование осознанию ценности выполненной работы. Достижение полного понимания изученного материала. Самооценка детей, сравнение результатов собственной деятельности с

			другими, осмысление результатов
Информация о домашнем задании, инструктаж по его применению	2 мин.	В этом домашнем задании вам предстоит обучить два вида композиций алгоритмов: бэггинг и бустинг. **Постановка задачи** Вам предлагается решить задачу бинарной классификации, а именно построить алгоритм, определяющий превысит ли средний заработок человека порог $50k. Каждый объект выборки — человек, для которого известны следующие признаки: • age • workclass • fnlwgt • education • education-num • marital-status • occupation • relationship • race • sex • capital-gain • capital-loss • hours-per-week	

Метрика качества
В качестве целевой метрики мы будем использовать ROC-AUC. Об этой метрике мы говорили в модуле о метриках. Как вы помните, для измерения ROC-AUC требуются вероятности принадлежности к классам. Для решающих деревьев вероятность принадлежности к классу вычисляется как доля объектов из обучающей выборки в соответствующем листе. Для алгоритма, который принимает решение взвешенным голосованием, вероятность вычисляется как среднее взвешенное значение вероятностей по всем алгоритмам в композиции.

Ход работы

- Первым делом вы произведете загрузку и обработку данных. В частности, вам необходимо будет закодировать категориальные признаки с помощью One-hot encoding.
- Сначала мы построим для нашей задачи обычный случайный лес и измерим его качество. Мы подберем оптимальный гиперпараметр "глубина дерева" для случайного леса.
- Далее мы обучим алгоритм градиентного бустинга с помощью библиотеки Catboost. Catboost --- это библиотека для градиентного бустинга, которая автоматически обрабатывает категориальные признаки. Поэтому для этого пункта вам нужно будет использовать не One-hot признаки, а изначальные категориальные признаки.

Оценивание

		В этом домашнем задании данные разделены на две части: • data_train.csv. Для этих данных вам известно значение целевой переменной. Эти данные вы будете использовать для обучения. • data_scoring.csv. На этих данных вы должны будете применить готовую модель, а затем сдать результаты в Stepik. Вам необходимо будет сдать результат работы двух моделей: случайного леса и градиентного бустинга.	

Рекомендуемые ресурсы для дополнительного изучения:

1. Научно-образовательный журнал для студентов и преподавателей «StudNet» №6/2020 [Электронный ресурс] – Режим доступа: https://cyberleninka.ru/article/n/podhody-k-postroeniyu-kompozitsiy-algoritmov-kak-sredstv-povysheniya-kachestva-prediktivnoy-modeli/viewer.
2. Ансамбли моделей: бэггинг, случайные леса, бустинг [Электронный ресурс] – Режим доступа: https://ranalytics.github.io/data-mining/044-Ensembles.html.
3. Стекинг (Stacking) и блендинг (Blending) [Электронный ресурс] – Режим доступа: https://dyakonov.org/2017/03/10/c%D1%82%D0%B5%D0%BA%D0%B8%D0%BD%D0%B3-stacking-%D0%B8-%D0%B1%D0%BB%D0%B5%D0%BD%D0%B4%D0%B8%D0%BD%D0%B3-blending/.