ТЕХНОЛОГИЧЕСКАЯ КАРТА ЗАНЯТИЯ

Тема занятия: Логическая регрессия

Аннотация к занятию: поговорим про обучение линейных алгоритмов с помощью градиентного спуска. Технология обучения, которую мы сейчас изучим, повсеместно используется во всём машинном обучении, не только при обучении линейных алгоритмов. В частности, обучение нейронных сетей происходит именно на основе градиентной оптимизации. Сначала мы обсудим нормализацию признаков: что это такое и с чем её едят. Затем перейдём к регуляризации: поговорим о проблеме мультиколлинеарности, которая и решается регуляризацией, обсудим способы регуляризации и особенности её применения.

Цель занятия: сформировать у обучающихся представление о логической регрессии. Познакомить с обучением линейных алгоритмов с помощью градиентного спуска. Обеспечить усвоение знаний о нормализации признаков, регуляризации. Обсудить проблему мультиколлинеарности, которая решается регуляризацией, обсудить способы регуляризации и особенности её применения.

Задачи занятия:

- познакомить с градиентным спуском для линейной регрессии;
- изучить стохастический градиентный спуск;
- разобрать градиентный спуск для логистической регрессии;
- узнать о нормализации признаков;
- узнать о регуляризации признаков.

Ход занятия

Этап занятия	Время	Деятельность педагога	Комментарии, рекомендации `для педагогов
Организационный этап	5 мин.	Дорогие друзья, здравствуйте. Продолжаем тему логистической регрессии.	Для справки: Логистическая регрессия — это метод классификации, используемый для прогнозирования значения категориальной зависимой переменной на основе связи с одной или несколькими независимыми переменными, которые предположительно имеют логистическое распределение. Если зависимое значение имеет только два возможных значения (успех или неудача), логистическая регрессия будет двоичной
Постановка цели и задач занятия. Мотивация учебной деятельности обучающихся	7 мин.	Сегодня мы обсудим, как вычислять вероятность принадлежности тому или иному классу, введём понятие логистической регрессии и запишем её в матричном виде.	Подробнее: https://ml-handbook.ru/chapters/model_evaluation/intro

		Вопрос для обсуждения Что такое классификация? **Возможный ответ обучающихся** Разделение по определённым характеристикам, признакам. Верно, пока что сконцентрируемся на случае обычной классификации объектов на, скажем, два класса. Рассмотрим задачу предсказания выживших пассажиров «Титаника». В этой задаче нам необходимо по признакам пассажира предсказать, спасся он или нет. В нашем распоряжении — обучающая выборка с размеченными объектами. Обозначим выживших пассажиров классом +1, не выживших — −1. Для простоты будем считать, что у каждого объекта есть всего два признака: возраст — действительное число, и пол — метка 0 или 1. Представим нашу выборку в координатах, соответствующих этим признакам. Объекты первого класса на картинке отмечены синим, объекты минус первого класса — красным. Наша цель — разбить плоскость на два множества: синее и красное.	

		Вопрос для обсуждения Как вы думаете, как будет работать алгоритм в данном примере? **Возможные ответы обучающихся** При попадании объекта в синее множество алгоритм решит, что объект относится к первому классу (то есть пассажир выжил), при попадании в красное — к минус первому (то есть пассажир погиб). Отлично. Верно.	
Изучение нового материала	55 мин.	**Вопрос для обсуждения** Ребята, давайте вспомним, в чём заключаются алгоритмы линейной и логистической регрессии? Задачи оптимизации, которые поставляют нам оптимальные веса алгоритмов? **Возможные ответы обучающихся** Для линейной регрессии решающее правило выглядит так: y с крышечкой от x равно сумме k_1x_1 + … + k_nx_n, то есть скалярное произведение вектора весов k и вектора признаков x. Задача оптимизации для линейной регрессии выглядит так: мы берём сумму квадратов разности правильного ответа y^i и ответа алгоритма (то есть этого самого скалярного произведения).	

		Оптимизируем функцию потерь по вектору параметров k. Теперь о логистической регрессии. Здесь в роли ответа выступает вероятность принадлежности к классу 1. То есть y с крышечкой от x равно сигмоиде от скалярного произведения k и x. Задача оптимизации выглядит так: минимизируем сумму логарифмов от единицы + экспоненты от минус y^i на скалярное произведение x^i и k. Как мы говорили ранее, задачи оптимизации мы будем решать с помощью градиентного спуска — разберём его алгоритм для линейной регрессии. Когда мы разбирали линейную регрессию, то обсуждали, что для решения задачи наименьших квадратов существует явная формула: оптимальный вектор k равен X транспонированное X в минус первой X транспонированное y. У этого подхода есть сложность: матрица X транспонированное X может оказаться необратимой. Справедливости ради, эта проблема может быть решена и без помощи градиентного спуска, но давайте попробуем найти минимальное значение k именно с помощью градиентного спуска. Итак, у нас есть задача оптимизации. Если мы хотим применить алгоритм градиентного спуска,	Для справки: Градиентный спуск — метод нахождения локального минимума или максимума функции с помощью движения вдоль градиента.

		нам необходимо первым делом вычислить градиент функции потерь по каждой переменной, по которой мы хотим оптимизировать. Мы оптимизируем вектор k, поэтому нам нужна производная по компоненте k_j. Давайте её считать. Запишем частную производную dL по dk_j. Первым делом поменяем местами производную и сумму. Осталось взять производную одного слагаемого этой суммы по k_j. То есть мы фиксируем все параметры, кроме k_j, и вычисляем производную функции от одной переменной. Можем воспользоваться правилом производной композиции функций, где внешняя функция — это квадрат, а внутренняя — это разность y^i — скалярное произведение k и x^i. Тогда производная равна производной внешней функции на производную внутренней функции. То есть два умножить на разность y^i минус скалярное произведение и умножить на множитель перед k_j, то есть на минус x с верхним индексом i и с нижним индексом j. Минус заносим в скобочку, получается сумма два умножить на x^i_j на разность скалярного произведения минус y. Мы вычислили производную по каждой компоненте. Теперь запишем градиент — это вектор из частных производных. Градиент также складывается из суммы ell слагаемых, где i-ое слагаемое — это вектор, составленный из	Для справки: https://radioprog.ru/post/773

		компонент 2x_j^i на разность скалярного произведения и игрека. Тогда это самое i-ое слагаемое можно записать просто как вектор x^i, домноженный на два и ещё домноженный на разность скалярного произведения и игрека, поскольку этот коэффициент будет одним и тем же для каждой компоненты вектора x^i. Поэтому в итоге градиент функции потерь по k равен удвоенной сумме x^i с коэффициентом разность скалярного произведения k, x^i минус y^i. Прекрасно, градиент функции потерь мы нашли. Можно переходить к алгоритму градиентного спуска для обучения линейной регрессии. Нам достаточно в обычный градиентный спуск подставить нужную нам функцию потерь и вычисленный на предыдущем слайде градиент. На вход алгоритму подаётся обучающая выборка X из ell элементов, а также вектор ответов y^1, ..., y^ell. Цель — найти оптимальный вектор параметров k. Сначала зафиксируем скорость обучения alpha — её называют learning rate (скорость обучения). Также мы должны выбрать изначальное приближение для параметра k — стартовую точку для градиентного спуска. Изначальное приближение можно выбрать любым, от этого мало что зависит. В процессе обучения мы на каждом шагу вычисляем градиент функции потерь при текущем значении весов и вычитаем вычисленный градиент, домноженный на альфа, из	Для справки: Стохастический градиентный спус к (часто сокращенно SGD) — это итерационный метод

		текущего вектора весов. Так мы потихоньку меняем значение вектора весов, постепенно приходя к оптимальному значению. Возникает вопрос: как долго мы должны это делать? В идеале — пока функция потерь не станет равной нулю на всех объектах. На практике этого достичь не получится, поэтому нам нужны другие критерии остановки. Мы обсудим дополнительные способы позже, а пока скажем, что делаем это, пока не надоест. Мы сформулировали алгоритм градиентного спуска. **Вопрос для обсуждения** Попытайтесь его проанализировать. **Возможные ответы обучающихся** Мы видим, что на каждом шаге алгоритм проходит по всей обучающей выборке и вычисляет градиент в виде суммы таких слагаемых, а затем вычитает сумму из текущего значения, тем самым настраивая коэффициенты регрессии. А если обучающая выборка достаточно большая, то расчёт градиента для одного шага будет занимать много времени, что нас не устраивает. **Вопрос для обсуждения** Давайте подумаем, как это исправить.	оптимизации целевой функции с подходящими свойствами гладкости (например, дифференцируемой или субдифференцируемой).

		Возможные ответы обучающихся Кажется, что использовать все элементы обучающей выборки на каждой итерации нет смысла. Ведь каждый конкретный элемент обучающей выборки потенциально может помочь нам немного изменить веса алгоритма. Что если на каждом шаге мы будем использовать только один объект? Модифицируем наш алгоритм. Этот модифицированный алгоритм называется стохастическим градиентным спуском. Слово «стохастический» означает «вероятностный». Алгоритм устроен почти так же, как и обычный градиентный спуск. Цикл, в котором мы проходимся по всем элементам обучающей выборки, называется одной эпохой обучения. На каждой эпохе мы по очереди проходимся по всем элементам обучающей выборки в случайном порядке. На каждом элементе мы вычисляем градиент функции потерь только по одному этому элементу, а затем сразу выполняем шаг градиентного спуска. Мы повторяем это для каждого элемента обучающей выборки. Тогда уже после первого прохода обучающей выборки полностью мы сделаем ell шагов градиентного спуска. Конечно, каждый из шагов будет учитывать только один элемент и поэтому	Подробнее: Функции потерь (Loss functions/objectives) позволяют вычислить ошибку алгоритма на каждом конкретном объекте. Выбираются обычно непрерывными и почти всюду дифференцируемыми для того чтобы применять градиентные методы.

		будет по-своему неточным. Но в совокупности все эти неточности компенсируют друг друга. Одной эпохи для обучения алгоритма нам, скорее всего, не хватит, поэтому мы должны провести несколько эпох. Чем больше эпох мы будем проводить, тем более точным получится итоговое приближение весов. Перечислим важные отличия стохастического градиентного спуска от обычного градиентного спуска. Во-первых, при стохастическом градиентном спуске процесс обучения идёт быстрее, так как алгоритм делает итерацию на каждом объекте. При этом, минимум ищется менее точно, так как на каждом шаге мы оптимизируем не функцию потерь по всей выборке, а смотрим только на лосс на одном объекте. Слово «лосс» — это калька от английского loss function и то же самое, что функция потерь. Чтобы использовать все преимущества стохастического градиентного спуска, но при этом нивелировать его недостатки, как правило рассматривают компромиссный вариант между обычным градиентным спуском и стохастическим. Компромисс заключается в том, чтобы случайным образом разбить обучающую выборку на небольшое кусочки фиксированного размера (например, 64). Кусочки часто называются	

		батчами, а их размер — batch size. По очереди проходимся по каждому батчу, считаем среднее значение градиента функции потерь по всем 64 элементам батча, а затем делаем шаг градиентного спуска по этому градиенту. Давайте проанализируем, к чему это приводит. С одной стороны, процесс обучения идёт гораздо быстрее, чем в случае обычного градиентного спуска, потому что на каждой эпохе мы обновляем градиент много раз. С другой стороны, процесс обучения будет довольно точным, потому что на каждом шаге градиент вычисляется не по одному элементу, а по 64, то есть по статистически репрезентативной части обучающей выборки. В машинном обучении почти всегда используют именно стохастический градиентный спуск с батчами. Иногда именно этот алгоритм называют просто стохастическим градиентным спуском. Наконец, поговорим о том, когда стоит останавливать обучение алгоритма с помощью стохастического градиентного спуска. Сколько эпох необходимо сделать? На этот счёт есть целые теории и много различных подходов. Мы укажем два из них. • Первый подход — это ограничивать количество эпох обучения. Например, если обучающая выборка небольшая, то можно просто запустить обучение на десять тысяч эпох	Для справки: Валидационная выборка (Выборка для разработки Dev (development) set) — выборка данных, которая используется для подбора параметров, выбора признаков и принятия других решений, касающихся обучения алгоритма. Её иногда называют удерживаемой выборкой для кросс-валидации (hold-out cross validation set).

		и не заботиться о том, когда мы будем останавливаться. Понятно, что десять тысяч — большое количество, и всё зависит от размеров обучающей выборки и того, сколько времени у нас есть на обучение. • Второй подход — после каждой эпохи измерять лосс на заранее отложенной валидационной выборке. Поскольку алгоритм не видит валидационную выборку при обучении, значение функции потерь на нём должно быть равно значению функции потерь на реальных данных. Когда лосс на валидационной выборке не уменьшается несколько эпох подряд, это значит, что модель больше не может вычленить новые значимые зависимости в данных. Можно заканчивать обучение. Часто эти две техники совмещают: обучают, пока лосс на валидационной выборке уменьшается, но не больше какого-то фиксированного количества эпох. Итак, мы обсудили стохастический градиентный спуск для линейной регрессии. Это универсальная техника обучения: её можно применить для обучения широкого класса алгоритмов, в частности, и для логистической регрессии. Давайте это проделаем. Выпишем задачу оптимизации для логистической регрессии. Функция потерь равна сумме по всем объектам обучающей выборки логарифма от 1	

		плюс e в степени минус y_i на скалярное произведение x_i с вектором весов k. Минимизируем эту функцию. Обратите внимание, что для этой функции потерь точку минимума в явном виде выписать не получится, в отличие от точки минимума в случае линейной регрессии. Поэтому единственное, что мы можем делать — это использовать градиентный спуск. Сначала считаем производную функции потерь по одному коэффициенту k_j. Как и в случае линейной регрессии, производная суммы равна сумме производных. Далее, берём производную от логарифма 1 + экспонента по правилу производной композиции. Производная внешней функции — это производная логарифма, то есть 1/x. Отсюда в знаменателе появляется 1 + e в степени −y^i на скалярное произведение. Теперь разберёмся с производной внутренней функции — 1 плюс экспонента. Производная от единицы — это ноль, поэтому нас интересует производная экспоненты. Чтобы вычислить производную экспоненты, снова применяем правило производной композиции. Внешняя функция — это собственно e^x. Внутренняя функция — это функция, линейная по k_j, поэтому её производная — это коэффициент при k_j, то есть это минус игрек i-ое умножить на x_j^i.	Для справки: http://www.machinelearning.ru/wiki/index.php?title=Мультиколлинеарность

		Получается, что в числителе стоит минус игрек i-ое на x_j^i и умножить на эту самую экспоненту. Теперь сокращаем числитель и знаменатель на эту самую экспоненту. Получаем итоговое выражение для нашей производной. Наконец, получаем выражение для градиента. Здесь выражение будет точно таким же, только у икса пропадает индекс j. Каждое слагаемое в получившемся градиенте — это вектор x^i умножить на некоторый коэффициент, который выделен на картинке. Градиент посчитан, теперь можем точно так же выписать алгоритм градиентного спуска. Внутри каждой эпохи мы вычисляем градиент по формуле, которую мы вывели на предыдущем слайде. Затем делаем шаг градиентного спуска. Ещё раз отметим, что в данной задаче градиентные методы — это единственная доступная нам опция, так как точное решение задачи оптимизации в явном виде выписать мы не сможем. Сначала мы рассмотрим нормализацию признаков: что это такое и с чем её едят. Затем перейдём к регуляризации: поговорим о проблеме мультиколлинеарности, обсудим способы регуляризации и особенности её применения.	Для справки: https://habr.com/ru/post/527334/

		Мы уже изучили базовые принципы построения алгоритмов линейной и логистической регрессии. Настало время поговорить о модификациях этих алгоритмов, которые в некоторых случаях весьма существенно помогают получить более качественные и стабильные модели. Первым делом поговорим о нормализации признаков. Грубо говоря, нормализация признаков — это приведение всех признаков к одному и тому же масштабу. Представьте, что в датасете при решении задачи машинного обучения есть признаки с очень разным масштабом: например, возраст клиента (который измеряется в годах и обычно не превосходит 100) и его зарплата, которая измеряется в рублях. **Вопрос для обсуждения** В каких условных единицах нам стоит измерять зарплату — в рублях или в тысячах рублей? **Ответы обучающихся** Для линейных алгоритмов лучше, если все признаки будут иметь один и тот же масштаб: то есть зарплату лучше измерять в тысячах рублей, потому что тогда большинство значений этого признака будет находится примерно в том же диапазоне, что и типичные значения возраста. Сейчас мы разберёмся, почему алгоритмы лучше работают, когда все признаки имеют один и тот же масштаб, и покажем, как автоматически	

		модифицировать признаки, приводя их к одному и тому же масштабу. Представим, что в датасете есть какой-то признак. Пусть x_1, x_2...x_ell — это значения одного и того же признака по всем элементам обучающей выборки. Наша цель — сделать этот признак центрированным и нормированным. Центрированный признак означает, что центр его распределения находится в нуле. Нормированный признак означает, что его среднее отклонение от нуля будет равно единице. Реализуем эту идею. Пусть a — это среднее значение признака — x_1 + x_2 + ... + x_ell делить на ell. Теперь вычислим величину сигма, равную следующему: это корень из среднего значения отклонения признака от величины a. Например, (x_1 - a)^2 — это квадрат первого элемента выборки от среднего значения этого признака. Квадраты отклонений суммируются и делятся на ell — получается среднее значение квадрата отклонения. Затем, чтобы привести значение отклонения в исходные единицы измерения, мы извлекаем из этой величины корень. Значение сигма называется средним квадратичным отклонением или стандартным отклонением. Тогда формула для нормировки признака будет такой: мы вычитаем среднее из икса и делим на стандартное отклонение.	

		Давайте эту идею поясним на картинке. Сверху вы видите гистограмму распределения исходного признака. Среднее значение признака равно 10, а стандартное отклонение относительно среднего равно 5 — это видно по длине красной стрелки. Снизу — гистограмма распределения того же признака после нормализации. Сама форма гистограммы не изменилась, но среднее значение теперь стало равным нулю, а стандартное отклонение стало равным единице. Таким образом, если нормировать все признаки, они приведутся к одному и тому же масштабу. Давайте кратко укажем, как нормализация помогает обучению линейных алгоритмов. Изначально может показаться, что нормализация признаков никак не влияет на работу самого алгоритма. Если мы делим признак на некоторую константу сигма, то можно вместо этого разделить весовой коэффициент k при этом признаке на ту же константу сигма. Иными словами, если в линейный алгоритм зарплата, выраженная в тысячах, входит с коэффициентом два, то это то же самое, как если бы зарплата, выраженная в единицах, входила бы с коэффициентом две тысячи. **Вопрос для обсуждения** Так зачем же на самом деле нужна нормализация? **Возможные ответы обучающихся**	

		• Первая причина — для лучшей оптимизации функции потерь. Пусть мы находим нужные нам веса с помощью градиентного спуска. Оказывается, если признаки имеют разный масштаб, то обучение весов, соответствующих этим признакам, тоже будет происходить с разной скоростью. Вычислительные алгоритмы лучше работают с нормированными признаками. • Вторая причина — это интерпретируемость весов модели. Представьте, что у признаков в линейном алгоритме одинаковый масштаб. Тогда, если коэффициент k у какого-то признака большой по модулю (неважно, положительный или отрицательный), то изменение этого признака будет сильно влиять на выход модели. Это означает, что важность данного признака при принятии решения высокая. Соответственно, если коэффициент k по модулю маленький, то важность этого признака мала и, возможно, его даже можно выбросить. Если бы изначально признаки не были нормированы, то сравнивать весовые коэффициенты при, скажем, тех же самых признаках «возраст» и «зарплата» мы бы не смогли. Итак, нормализация помогает сделать правильные выводы об устройстве модели и о том, какие признаки она выделяет как наиболее важные. Важность признаков часто используется в индустрии. Мы поняли, что от нормализации	

		признаков только плюсы и никаких минусов. Поэтому предлагаю вам делать её всегда. Перейдём к ещё одной важной теме — регуляризации. Если кратко, регуляризация — это средство сделать модель машинного обучения более стабильной. Начнём с описания проблемы — явления мультиколлинеарности в признаках. Для примера рассмотрим вот такой игрушечный датасет. В нём 4 объекта — x^1, x^2, x^3, x^4 и 3 признака — F_1, F_2, F_3. В матрице объекты-признаки есть следующие элементы: 2, 3, 5; −1, 2, 1; 3, −2, −1 и 0, 1, 1. Эту матрицу мы обозначим через X. Числа подобраны так, чтобы столбцы матрицы X были линейно зависимы. Если мы внимательно посмотрим на таблицу, то увидим, что третий столбец равен сумме первых двух. Это означает, что можно написать равенство X умножить матрично на beta равно нулю, где beta — это вектор 1, 1, −1. В самом деле, X умножить на beta — это как раз сумма первых двух столбцов минус третий столбец, а это равно нулю. Пусть мы обучили алгоритм линейной регрессии на этих данных (здесь не так важно, какие конкретно у нас были игреки для обучения). Обученный вектор весов получился таким: k = 2, −3, 1. Тогда предсказания этого алгоритма на обучающей выборке — это матрица X умножить	

		матрично на вектор k. Если посчитать, получится вектор 0, −7, 11, −2. Заметим следующее: для каждого конкретного вектора весов k существует ещё целое бесконечное множество векторов весов, при которых алгоритм на обучающей выборке работает точно так же, как и если бы вектор весов был равен k. Пусть вместо k вектор весов равен k + c умножить на beta. Здесь beta — это тот самый вектор 1, 1, −1. Тогда посмотрим на работу алгоритма с этими весами. Получим X умножить на k + c beta равно... раскрываем скобки по дистрибутивности перемножения матриц, получаем X умножить на k плюс X умножить на c beta, то есть просто cX умножить на beta. Теперь, поскольку X умножить на beta равно нулю, то всё это выражение просто равно X умножить на k. Итак, для вектора весов k + c beta результат работы алгоритма равен тому же самому вектору 0, −7, 11, −2, как и для вектора весов k. Чем же тогда вектор k лучше вектора k + c beta, если линейная регрессия в том и в другом случае работает на наших данных одинаково? Подчёркиваю, сами алгоритмы сильно различаются, но на обучающей выборке. На данных, которые есть в нашем распоряжении, они работают одинаково. Значит, ни один из них не лучше и не хуже другого. Кажется, сложностей нет. Но представьте, каково в этой ситуации дата сайентисту. В понедельник	

		он обучил модель на этих данных, она получилась одна. А во вторник он случайно стёр модель и обучил её заново, она получилась уже другая. Да и вообще, представьте, что, скажем, beta равно миллиону. Тогда веса алгоритма k + c beta получились бы огромными — тоже порядка миллиона. И любая маленькая погрешность в тестовых данных будет домножаться на эти неадекватно большие коэффициенты, что будет сразу же приводить к неадекватным ответам. Действительно, мультиколлинеарность признаков влечёт нестабильность весов. **Это ведёт к нескольким проблемам.** • Во-первых, оптимальное решение является нестабильным и, как следствие, невоспроизводимым и непредсказуемым. • Во-вторых, веса самого алгоритма могут произвольно получаться очень большими, что ведёт к неустойчивости к выбросам и даже самым маленьким изменениям в данных. Наконец, если посмотреть на алгоритм линейной регрессии с точки зрения явной формулы, то явную формулу для вычисления весов просто нельзя посчитать! Если столбцы матрицы X линейно зависимы, то матрица X транспонированное X будет вырожденной и, следовательно, необратимой. **Вывод:** мультиколлинеарность — это не очень хорошо, и с ним нужно как-то бороться.	Для справки: Ридж-регрессия или гребневая регрессия (англ. ridge regression) — это один из методов понижения размерности. Часто его применяют для борьбы с переизбыточностью данных, когда независимые переменные коррелируют друг с другом (т.е. имеет место мультиколлинеарность).

		Давайте поборемся. Как мы уже упомянули, наибольшие проблемы с алгоритмом проявляются тогда, когда веса оказываются слишком большими по модулю. Решение: попробовать как-то объяснить модели, что она не должна делать веса слишком большими. **Вопрос для обсуждения** Как это сделать? **Ответы обучающихся** Штрафовать модель за слишком большие веса. Пусть у нас есть задача линейной регрессии. Давайте добавим в нашу функцию потерь MSE дополнительное слагаемое, которое называется регуляризационным слагаемым. А именно сумму квадратов компонент вектора, домноженную на некоторую заранее зафиксированную константу C > 0. И теперь уже будем минимизировать не изначальную функцию потерь, а новую, регуляризованную. Тогда видно, что чем больше веса нашего алгоритма, тем больше значение L(k). Это значит, что в процессе минимизации алгоритм должен сам выбрать наиболее стабильную модель из тех, которые хорошо предсказывают ответы. Понятно, что без регуляризации решение на обучающей выборке будет давать результаты лучше, чем с регуляризацией, но регуляризация даёт алгоритму необходимую стабильность. Такой вид регуляризации называется L_2-регуляризацией (двойка — потому что	

		штрафуется сумма квадратов весов), а сам алгоритм называется ridge-регрессией (или гребневой регрессией). Второй способ регуляризации — это L_1-регуляризация или, как ещё её называют, lasso-регуляризация. Она отличается от L_2-регуляризации тем, что здесь регуляризационное слагаемое равно не сумме квадратов компонент, а сумме модулей компонент. Ясно, что такую регуляризованную версию линейной регрессии можно в точности так же обучать с помощью градиентного спуска, как и изначальную версию. Поэтому сложность обучения алгоритма практически не меняется. Константа C при регуляризационном слагаемом является гиперпараметром алгоритма и фиксируется перед обучением. Проанализируем отличия ridge-регрессии от lasso-регрессии. Для этого взглянем на график, соответствующий ridge-регрессии. Здесь по оси абсцисс отложены различные значения константы C. Для каждого из значений мы обучаем алгоритм линейной регрессии с L_2 регуляризацией и с константой регуляризации C. Для каждого признака у нас получается некоторый вес. А теперь давайте посмотрим на значение веса при одном и том же признаке, но при РАЗЛИЧНЫХ значениях	

		константы C. Как раз одна кривая на графике соответствует поведению одного веса при изменении C. У нас 14 признаков. Таким образом, мы строим 14 кривых. Здесь есть несколько тенденций. Во-первых, с ростом коэффициента штрафа C оптимальные веса уменьшаются — это в общем-то неудивительно. Во-вторых, они уменьшаются довольно плавно с ростом C. Никаких резких скачков не происходит. А теперь нарисуем такой же график для lasso-регрессии, то есть L_1. Здесь мы видим, что веса тоже уменьшаются с ростом C, но уменьшение происходит более резко и сразу к нулю: один раз зайдя в ноль, значение веса уже из нуля не уходит. То есть алгоритм действует так: с ростом константы C он выкидывает всё больше и больше ненужных признаков, оставляя в деле только самые важные. **Вопрос для обсуждения** Кто сможет сделать вывод? **Ответы обучающихся** При L_2-регуляризации уменьшение весов с ростом константы C более плавное, что делает этот выбор более предпочтительным при построении линейной модели. Но L_1-регуляризация тоже полезна: её можно использовать для построения модели из небольшого количества признаков. Ещё lasso-регрессию можно использовать в качестве	

		промежуточной модели для отбора наиболее важных признаков, чтобы на следующем этапе построить более сложную и хитрую модель только из признаков, наиболее важных в данной задаче. Напоследок скажу, что регуляризация используется не только в линейной, но и в логистической регрессии. Здесь точно так же можно добавить к функции потерь регуляризационное слагаемое, например, l_2-регуляризацию, и в результате логрегрессия получится более стабильной. Регуляризация — это важнейшая техника в машинном обучении, которая призвана повышать стабильность модели. Само слово регуляризация означает, что модель после регуляризации оказывается более регулярной — то есть стабильной или, если угодно, робастной — устойчивой к аномалиям и выбросам в данных. Можно рассматривать регуляризацию и как один из способов борьбы с переобучением. Регуляризационное слагаемое легко может быть добавлено при обучении почти любого алгоритма, основанного на градиентном спуске для оптимизации функции потерь. Оно не усложняет процесс обучения, и поэтому рекомендуется буквально всегда. В частности, в библиотеке Sklearn модель логистической регрессии по умолчанию строится с регуляризационным слагаемым.	

		Вопрос для обсуждения Что вы сегодня узнали? **Ответы обучающихся** Мы рассмотрели обучение линейных алгоритмов с помощью градиентного спуска. На каждом шаге алгоритма мы вычисляем градиент функции потерь, затем вычитаем его из вектора весов, предварительно домножив на коэффициент альфа — скорость обучения (или learning rate). Также мы изучили стохастический градиентный спуск. Здесь мы на каждом шаге вычисляем градиент функции потерь не по всей выборке, а по отдельному батчу. За счёт стохастического градиентного спуска обучение идёт быстрее и почти не теряет в качестве. Регуляризация и нормализация — это полезные техники машинного обучения, позволяющие улучшить качество работы и стабильность алгоритмов. Регуляризация лучше работает именно с нормализованными признаками, поэтому их предлагается использовать вместе. Для добавления этих техник в модель вам необходимо сначала нормализовать признаки, затем записать старую функцию потерь и добавить к ней регуляризационное слагаемое. Далее модель можно обучать стохастическим градиентным спуском	

Закрепление изученного материала	10 мин.	**Вопросы для обсуждения** ● Что такое градиентный спуск для линейной регрессии? ● Расскажите о стохастическом градиентном спуске. ● Что такое градиентный спуск для логистической регрессии? ● Что такое нормализация признаков? ● Что такое регуляризация признаков?	Педагог организует беседу по вопросам
Этап подведения итогов занятия (рефлексия)	8 мин.	**Вопросы для обсуждения** ● Чему я научился? ● С какими трудностями я столкнулся? ● Каких знаний мне не хватает для более глубокого понимания изученного материала? ● Достиг ли я поставленных целей и задач?	Педагог способствует размышлению обучающихся над вопросами
Информация о домашнем задании, инструктаж по его применению	5 мин.	Дома повторите основные определения, просмотрите математические действия, которые мы с вами сегодня разбирали	

Рекомендуемые ресурсы для дополнительного изучения:

1. Линейная регрессия и градиентный спуск. [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/471458/.
2. Метод градиентного спуска. [Электронный ресурс] – Режим доступа: https://www.dmitrymakarov.ru/opt/gradient-02/.
3. Понятие скорости обучения в нейронных сетях. [Электронный ресурс] – Режим доступа: https://radioprog.ru/post/773.
4. Функция потерь (Loss Function). [Электронный ресурс] – Режим доступа: https://www.helenkapatsa.ru/funktsiia-potieri/.
5. Умная нормализация данных. [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/527334/.
6. Мультиколлинеарность. [Электронный ресурс] – Режим доступа: http://www.machinelearning.ru/wiki/index.php?title=Мультиколлинеарность.