**Опеределение** //слайд 2 и слайд 3

**Машинное обучение** – это наука о разработке алгоритмов и статистических моделей, которые компьютерные системы используют для выполнения задач без явных инструкций, полагаясь вместо этого на шаблоны и логические выводы.

Компьютерные системы используют алгоритмы машинного обучения для обработки больших объемов статистических данных и выявления шаблонов данных. Таким образом, системы могут более точно прогнозировать результаты на основе заданного набора входных данных. Например, специалисты по работе с данными могут обучить медицинское приложение диагностировать рак по рентгеновским изображениям, сохраняя миллионы отсканированных изображений и соответствующие диагнозы.

**Различие между AI и MO** //слайд 4 и слайд 5

*Искусственный интеллект (ИИ или AI агнл.) — это попытка копирования мыслительного процесса человека вычислительной машиной.*

Когда исследования в области Искусственного интеллекта только начинались, учёные пытались копировать поведение человеческого интеллекта строго в определённых условиях, то есть затачивать его под решение определённых задач. Например, для того, чтобы машина могла играть в игры. Они устанавливали ряд правил, которым вычислительная машина должна была следовать. У компьютера был перечень возможных действий, и он принимал решения, исходя из правил и ограничений, заданных на этапе разработки.

Машинное обучение (МО или ML англ.), означает возможность машины обучаться, обрабатывая большие наборы информации вместо четко прописанных правил.

**Контролируемое обучение vs Неконтролируемое обучение**

//слайд 6

Неконтролируемое обучение — это задача которая состоит в обучении ИИ с используя не структурированные данные.

При неконтролируемом обучении, алгоритм строится таким образом чтобы предсказывать ответы без маркировки человеком (или даже без вопросов). Вместо определения маркировки или результата, алгоритмы неконтролируемого обучения используют большие массивы данных и вычислительные мощности для выявления ранее не известных взаимоотношений.

**Пример** искусственного интеллекта, использующего неконтролируемое машинное обучение — робот-предсказатель поведения клиентов интернет магазина. Он обучается, не используя заранее известные входные и выходные данные. Вместо этого он должен самостоятельно классифицировать входные данные. Алгоритм должен определить и сообщить вам, какой тип пользователей предпочитает какие продукты.

//слайд 7

Контролируемое обучение использует маркированные наборы данных, которые состоят из входных данных и ожидаемых результатов.

**В качестве примера контролируемого обучения** можно рассмотреть некое интеллектуальное приложение для торговли недвижимостью. Его можно натренировать с помощью вектора признаков, включающего размер, количество комнат, и год постройки для набора домов. Человек должен присвоить каждому дому метку с правильной ценой дома, базируясь на этих факторах. Анализируя эти данные, умное приложение должно натренироваться, чтобы ответить на вопрос «Сколько денег я смогу получить за этот дом?».

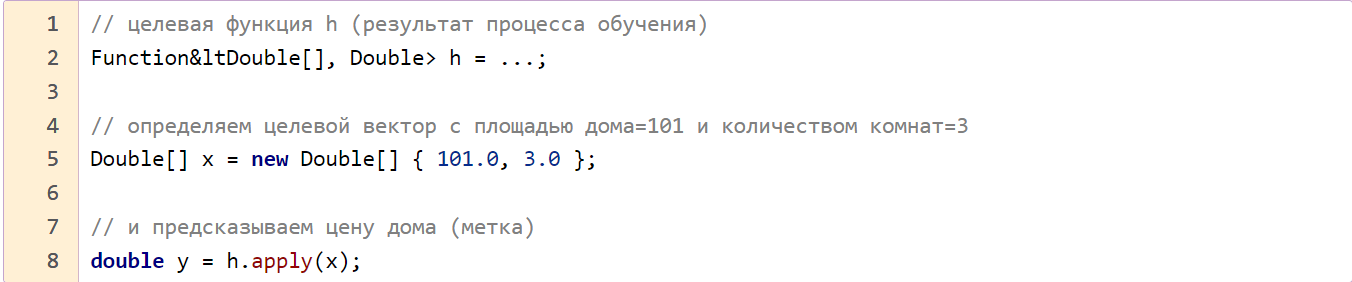
После того как процесс подготовки закончен, новые входные данные уже больше не маркируются. Машина должна быть способна правильно отвечать на вопросы, даже для не известных, не промаркированных векторов признаков.

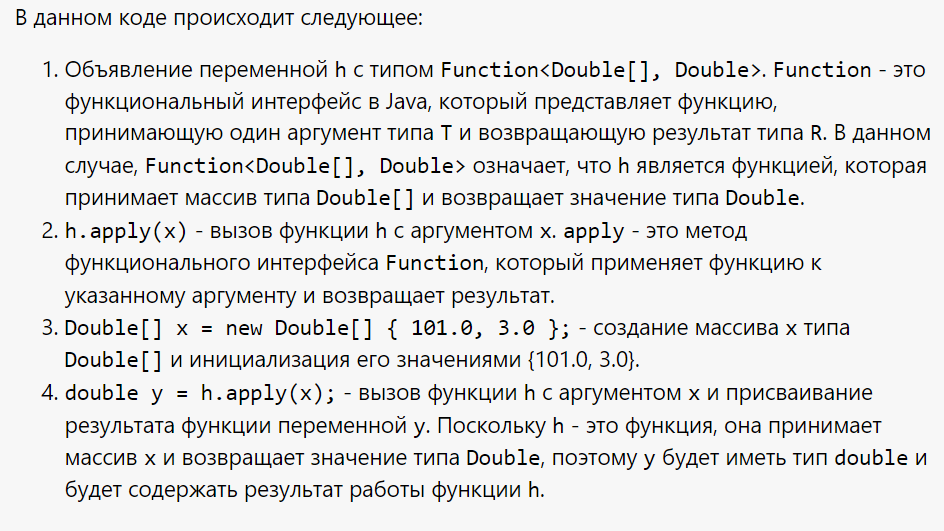
Любое машинное обучение базируется на данных. Для проекта по контролируемому машинному обучению, нужно отметить маркерами данные таким образом, чтобы получить осмысленные ответы на задаваемый вопрос.

//слайд 8

В Таблице, каждая запись информации о доме имеет метку «цена дома». Выявляя взаимосвязь между данными записей и ценой дома, алгоритм в итоге должен быть способным предсказать рыночную цену для домов, не входящих в данный список. (площадь дома указана в квадратных метрах, а стоимость дома в евро).

//слайд 9, 10, 11





Целью машинного обучения является определение целевой функции, которая будет работать максимально точно при неизвестных входных параметров. В машинном обучении, целевая функция (hθ) иногда называется моделью. Эта модель является результатом процесса обучения.

//слайд 12

**Линейная регрессия**

Чтобы научить машину «думать», сначала нужно выбрать алгоритм обучения, который вы будете использовать. Например, линейную регрессию. Это – один из простейших и самых популярных алгоритмов контролируемого машинного обучения. Алгоритм предполагает что отношение между входными признаками и маркерами результата – линейно. Общая функция линейной регрессии, приведенная ниже, возвращает предсказанное значение путем суммирования всех элементов вектора признаков умноженных на параметр θ (тета). Этот параметр используется в процессе обучения для адаптации или «подстройки» функции регрессии на основе тренировочных данных.

Обратите внимание, что признак x0 является постоянным элементом сдвига и имеет значение 1 для вычислительных целях.

//Объяснение задачки:

В результате индекс значимых параметров таких, как площадь дома, начинается с x1. Так, если x1 присвоено первое значение вектора признаков (площадь дома), то x2 будет принимать следующее значение (количество комнат) и так далее.

//слайд 13



Конструктор LinearRegressionFunction инициализирует массив thetaVector с переданными значениями.

Метод apply вычисляет прогнозное значение на основе весов thetaVector и входных признаков featureVector.

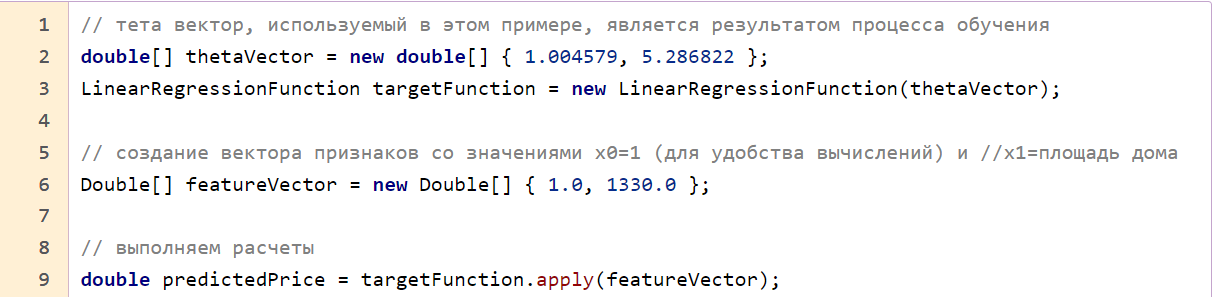
Метод getThetas возвращает копию массива thetaVector, содержащего веса модели.

Таким образом, класс LinearRegressionFunction представляет функцию линейной регрессии, которая принимает входные признаки и возвращает прогнозное значение на основе весов модели.

Чтобы создать новый экземпляр LinearRegressionFunction, нужно задать параметр θ. Этот параметр или вектор используется для адаптации общей функции линейной регрессии к лежащим в основе тренировочных данных. Параметр θ, используемый в программе, будет настроен в процессе обучения, базируясь на тренировочных примерах. Качество обученной целевой функции будет зависеть от качества подготовленных для обучения данных.

//слайд 14

В примере мы используем LinearRegressionFunction для иллюстрации предсказания цены, базируясь на размере дома. Принимая во внимание то, что x0 должен быть константой со значением 1.0, целевая функция инициализируется, используя два параметра θ, при этом они являются результатом процесса обучения. После создания нового примера, цена дома площадью 1330 квадратных метров будет предсказываться как показано в коде:



//слайд 15

//тут представлен график синей линией

Сейчас график предсказаний выглядит достаточно хорошо. Координаты графика (положение и наклон) определяются вектором θ { 1.004579, 5.286822 }. Но как определить, что именно этот θ-вектор лучше всего подходит для вашего применения? Будет ли функция соответствовать лучше, если вы измените первый или может быть второй параметр? Для определения, наиболее подходящего тета вектора вам нужна функция стоимости, которая будет оценивать насколько хорошо целевая функция справляется с этой задачей.

//слайд 16

//слайд 15

**Про функцию стоимости**

Для определения стоимости целевой функции, показанной выше, необходимо рассчитать квадратичную ошибку каждого примера дома (i). Ошибка – расстояние между расчетным значением у и настоящим значением y дома из примера i.

Например, реальная цена дома площадью 1330 = 6,500,000 €. А отличие предсказанной цены дома обученной целевой функцией составляет 7,032,478 €: разница (или ошибка) равна 532,478 €. Вы также можете увидеть эту разницу на графике выше. Разница (или ошибка) показана в виде вертикальных пунктирных красных линий для каждой тренировочной пары цена – площадь. //это на графике //слайд 16

//Объяснение реализации функции стоимости на JAVA:

//слайд 17

Далее приведена простая реализация на Java функции стоимости, принимающей на вход целевую функцию, список тренировочных данных, и метки связанные с ними. Значения предсказаний будут вычисляться в цикле, и ошибка будет вычисляться вычитанием реального значения цены (взятого из метки). Позже квадрат ошибок будет просуммирован и значение ошибки будет рассчитано. Стоимость будет возвращена как значение типа double:

//слайд 18

**Градиентный спуск**

Градиентный спуск минимизирует функцию стоимости. Это значит, что он используется для поиска параметров тета, которые имеют минимальную стоимость (J(θ)) на основе тренировочных данных.

//слайд 19

Так вот, параметры вектора тета будут улучшаться с каждой итерацией алгоритма. Коэффициент обучения α задает количество вычислений на каждой итерации. Эти вычисления можно проводить, пока не найдены «хорошие» значения тета.

//слайд 20

Для примера, функция линейной регрессии ниже имеет три параметра тета:

Данный код представляет собой метод train, который используется для обучения модели линейной регрессии. Вот объяснение построчно:

java

Copy

public static LinearRegressionFunction train(LinearRegressionFunction targetFunction,

List<Double[]> dataset,

List<Double> labels,

double alpha) {

Метод train принимает объект targetFunction типа LinearRegressionFunction, список dataset с входными признаками, список labels с соответствующими метками и значение alpha, которое является скоростью обучения.

java

Copy

int m = dataset.size();

Переменная m устанавливается равной размеру списка dataset, что представляет количество обучающих примеров.

java

Copy

double[] thetaVector = targetFunction.getThetas();

double[] newThetaVector = new double[thetaVector.length];

Получаем текущие значения весов модели thetaVector из targetFunction и создаем новый массив newThetaVector для хранения обновленных значений весов.

java

Copy

for (int j = 0; j < thetaVector.length; j++) {

double sumErrors = 0;

for (int i = 0; i < m; i++) {

Double[] featureVector = dataset.get(i);

double error = targetFunction.apply(featureVector) - labels.get(i);

sumErrors += error \* featureVector[j];

}

double gradient = (1.0 / m) \* sumErrors;

newThetaVector[j] = thetaVector[j] - alpha \* gradient;

}

В этом цикле происходит обновление значений весов модели. Для каждого элемента вектора thetaVector вычисляется сумма ошибок, умноженных на соответствующие признаки. Затем вычисляется градиент, который представляет собой среднюю ошибку, умноженную на скорость обучения alpha. Новые значения весов сохраняются в newThetaVector.

java

Copy

return new LinearRegressionFunction(newThetaVector);

Возвращается новый объект LinearRegressionFunction, созданный с использованием обновленных значений весов newThetaVector.

Таким образом, метод train выполняет одну итерацию обучения модели линейной регрессии, вычисляя градиенты и обновляя значения весов.

//объяснение кода реализации алгоритма градиентного спуска

Тета для функции регрессии будут обучены с использованием тренировочных данных, данных маркеров, коэффициента обучения (α). Результатом будет улучшенная целевая функция, использующая параметры тета. Метод train() будет вызываться снова и снова, и передавать новую целевую функцию и новые параметры тета из предыдущих вычислений. И эти вызовы будут повторяться, пока настроенная целевая функция не достигнет плато минимума:

// слайд 21

// потом идет анализ результатов отработки программы

Чтобы убедиться, что стоимость постоянно уменьшается, можно запускать функцию стоимости J(θ) на исполнение после каждого шага обучения. После каждой итерации стоимость должна уменьшаться. Если этого не происходит, это значит, что значение коэффициента обучения слишком большое и алгоритм просто проскочил минимальное значение. В таком случае алгоритм градиентного спада терпит неудачу.

Графики ниже демонстрируют целевую функцию, использующую новые, вычисленные, тета-параметры, начинающиеся со стартового тета-вектора {1.0, 1.0}.

Левая колонка показывает график функции предсказания после 50 повторений; средняя колонка после 200 повторений; и правая колонка после 1000 повторений.

Из них видно, что цена уменьшается после каждой итерации, и новая целевая функция соответствует все лучше и лучше. После 500-600 повторений тета-параметры больше существенно не меняются, и цена достигает стабильного «плато». После этого точность целевой функции улучшить таким способом не получится.

//посмотрели на еще 6 графиков, проанализировали и сделали вывод

В таком случае, несмотря на то, что стоимость больше значительно не уменьшается после 500-600 итераций, целевая функция все еще не оптимальна. Это свидетельствует о несоответствии. В машинном обучении термин «несоответствие» используется для обозначения того что алгоритм обучения не находит основные тенденции данных. Если обратиться к реальному опыту, вполне вероятно ожидать уменьшения цены за квадратный метр для бОльших владений. Отсюда мы можем сделать вывод, что модель, использованная для процесса обучения целевой функции, не соответствует данным в достаточной мере. Несоответствие часто связанно с чрезмерным упрощением модели. Так произошло и в нашем случае, целевая функция слишком простая, и для анализа использует единственный параметр — площадь дома. Только вот этой информации недостаточно для точного предсказания цены дома.

//решение проблемы:

// слайд 22

### Добавление признаков и их масштабирование

Если вы обнаружили что ваша целевая функция не соответствует проблеме, которую вы пытаетесь решить, её нужно подкорректировать. Распространенный способ корректировки несоответствия — добавление дополнительных признаков в вектор признаков.

В примере с ценой дома, можно добавить такие характеристики, как количество комнат или возраст дома. То есть вместо использования вектора с одним значением признака {size} для описания дома, можно использовать вектор с несколькими значениями, например, {size, number-of-rooms, age}.

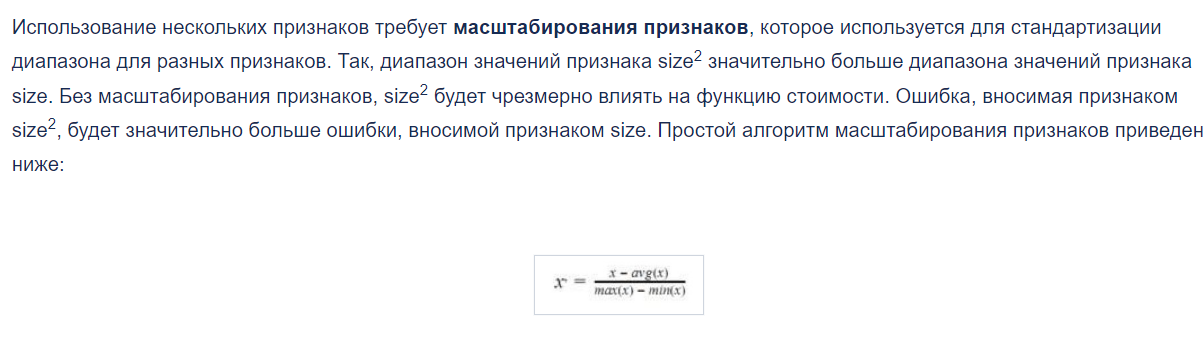
В некоторых случаях количества признаков в доступных тренировочных данных не хватает. Тогда стоит попробовать применить полиномиальные признаки, которые вычисляются с использованием существующих.

// слайд 23

Например, вы имеете возможность расширить целевую функцию определения цены дома таким образом, чтобы она включала вычисляемый признак квадратных метров (x2):

//тут расширили целевую функцию

// слайд 22



Этот алгоритм реализован в классе FeaturesScaling в коде примера ниже. Класс FeaturesScaling представляет промышленный метод для создания функции масштабирования подстраиваемой на тренировочных данных. Внутри экземпляры тренировочных данных используются для вычисления среднего, минимального и максимального значений.

// слайд 24

Результирующая функция использует вектор признаков и производит новый с отмасштабированными признаками. Масштабирование признаков необходимо как для процесса обучения, так и для процесса предсказания, как показано ниже:

//реализация кода на слайде

//предостережение насчет программы

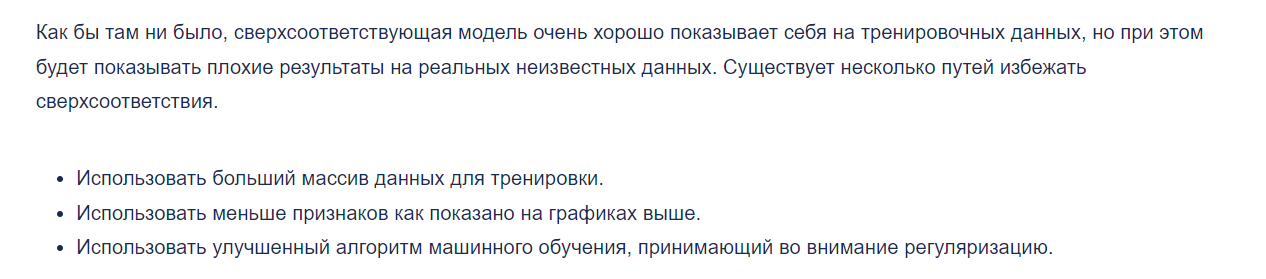
С добавлением всё большего и большего количества признаков, становится заметен рост соответствия целевой функции, однако будьте осторожны. Если вы зайдете слишком далеко и добавите слишком много признаков, вы можете в результате поучить целевую функцию, которая сверхсоответствует.

// слайд 25

Сверхсоответствие возникает тогда, когда целевая функция или модель соответствует тренировочным данным слишком хорошо, настолько, что захватывает шум или случайные отклонения в тренировочных данных. Пример сверхсоответствия приведен на графике:

// слайд 26

// слайд 27, 28

 Если алгоритм предсказания сверхсоответствует тренировочным данным, необходимо исключить признаки, которые не приносят пользы для его точности. Сложность составляет поиск признаков, которые существеннее других влияют на точность предсказания. Как показано на графиках, сверхсоответствие можено определить визуально с помощью графиков. Это хорошо работает для графиков с 2 или 3 координатами, становится трудно построить и оценить график если вы используете больше чем 2 признака. В перекрестной проверке вы перепроверяете модели после обучения с использованием данных не известных алгоритму после окончания процесса обучения.

// слайд 29

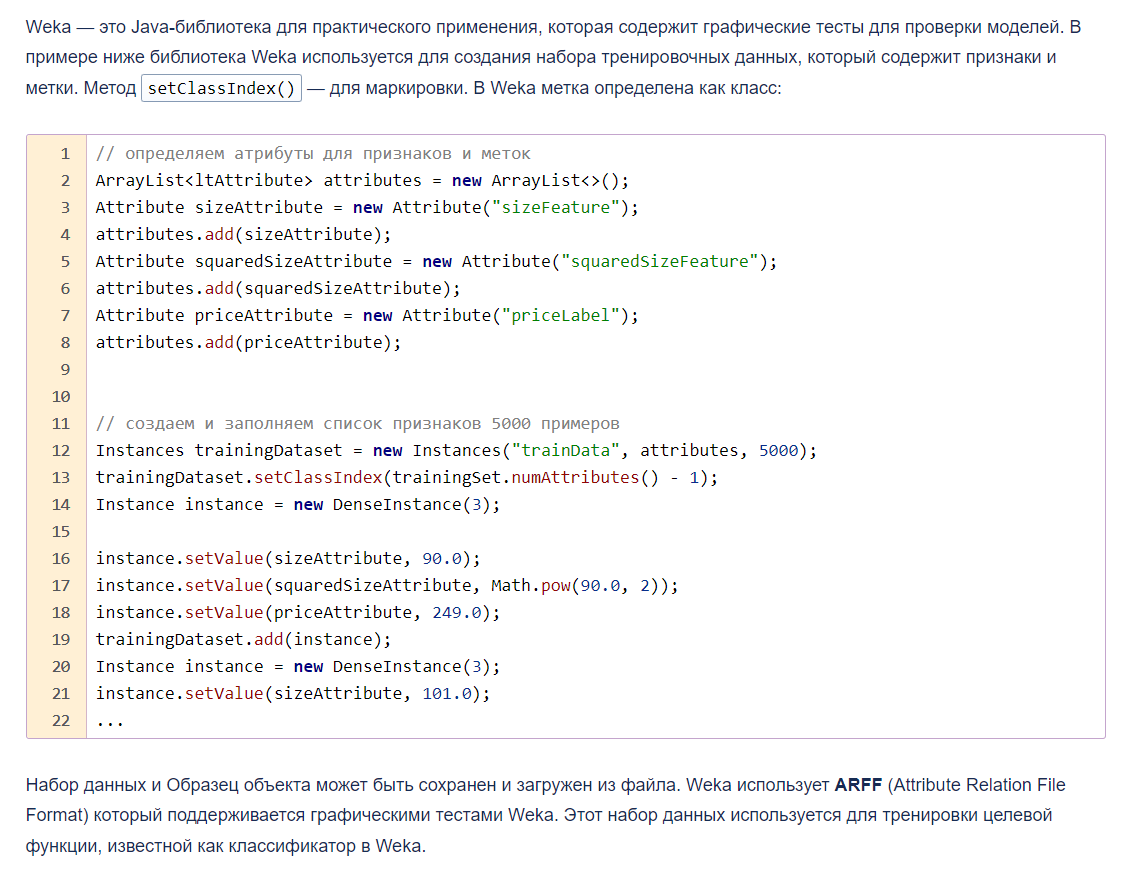
Доступные маркированные данные должны быть разбиты на 3 набора:

* тренировочные данные;
* проверочные данные;
* тестовые данные.

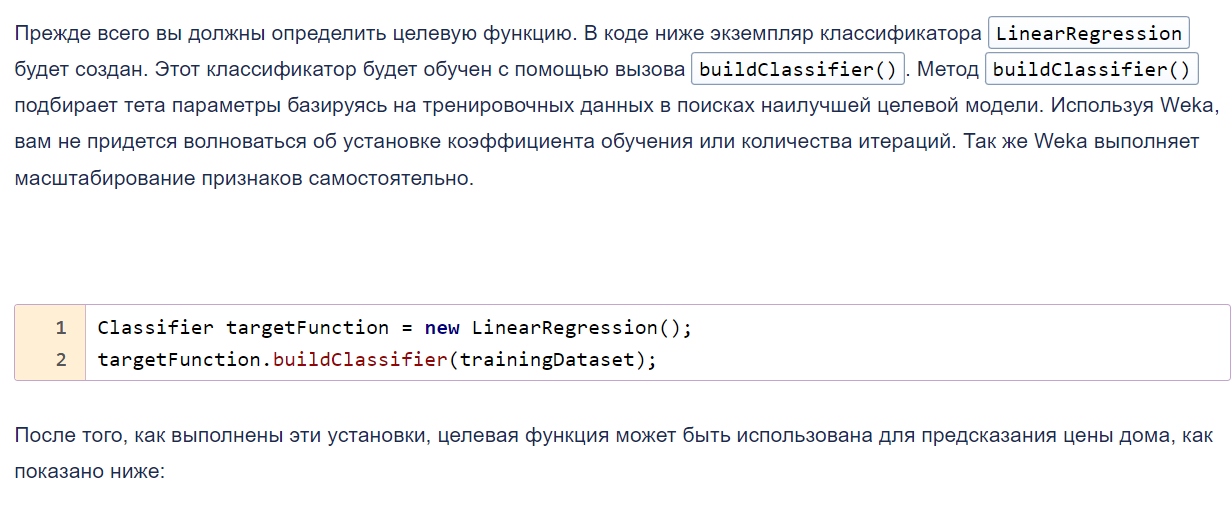
В таком случае 60 процентов промаркированных записей, характеризующих дома, должны быть использованы в процессе обучения вариантов целевого алгоритма. После процесса обучения, половину оставшихся данных (не использованных ранее), нужно использовать для проверки того факта, что обученный целевой алгоритм работает хорошо с неизвестными данными. Как правило, алгоритм, который показывает лучшие результаты по сравнению с другими, и выбирается для использования. Оставшиеся данные используются для вычисления величины ошибки для окончательно выбранной модели. Существуют и другие техники перекрестной проверки, например, как **k-fold**.

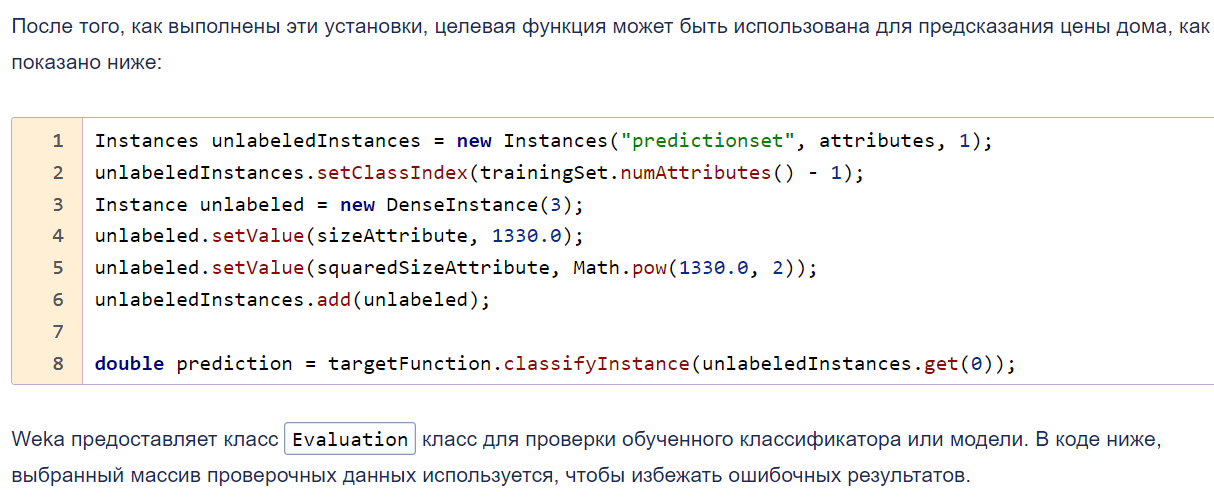
// слайд 30

// слайд 31

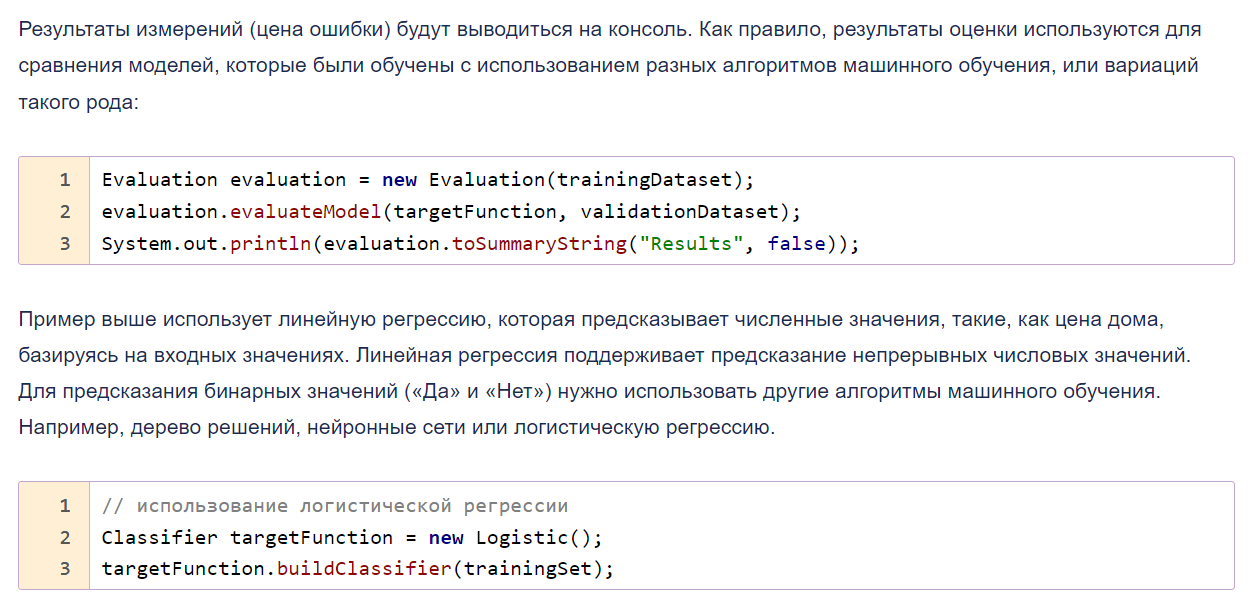


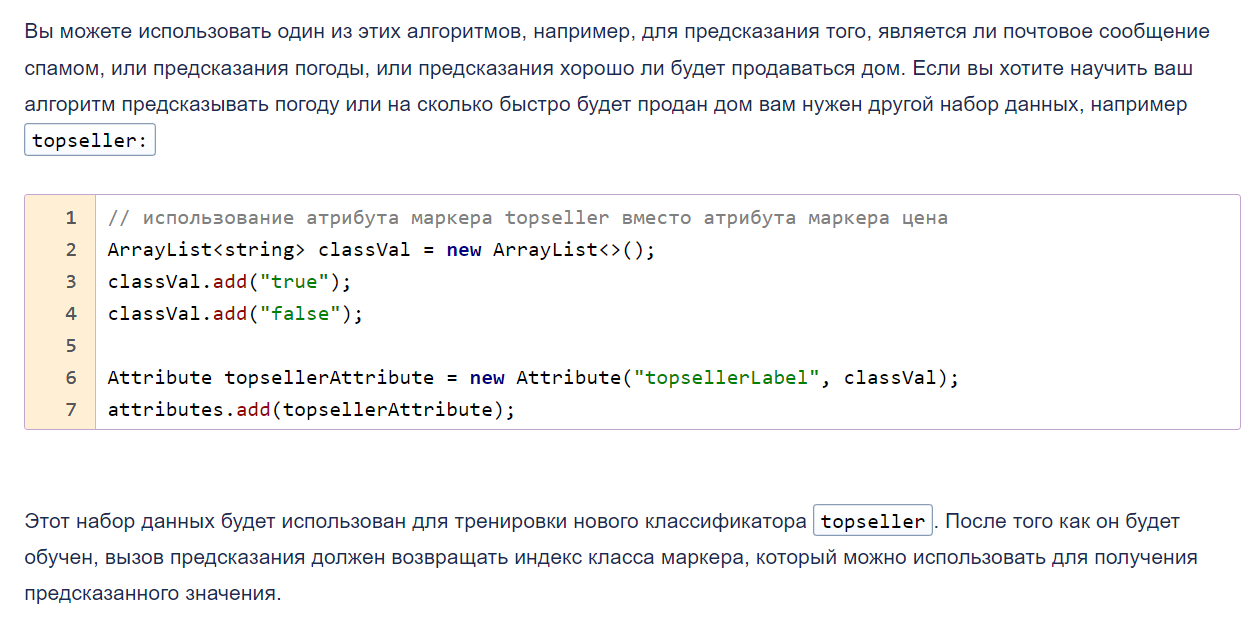
//слайд 33



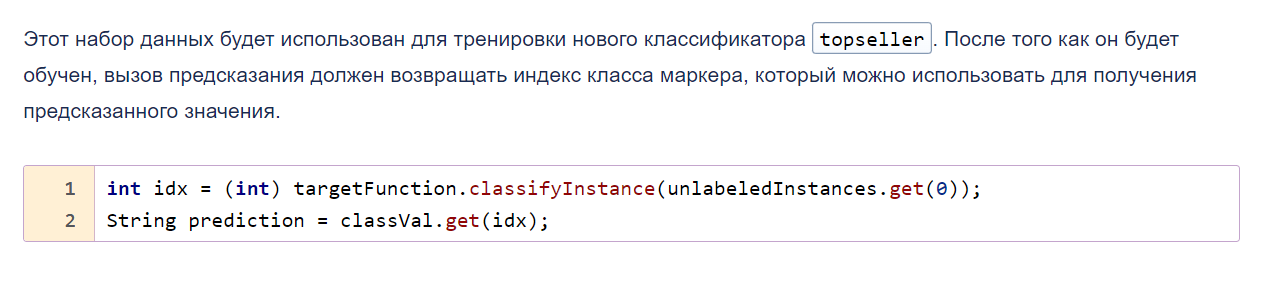


//слайд 33





//слайд 34



//слайд 35

//слайд 36

