Лекция 1. Базовые понятия машинного обучения. Основные инструменты машинного обучения.

Машинное обучение – это наука о том, как научить машину самостоятельно решать задачи. Цель машинного обучения – научить машину (точнее, программу) решать задачу, предъявив ей несколько примеров (с правильными и неправильными решениями). Предположим, что каждое утро, включив компьютер, вы делаете одно и то же: сортируете пришедшую почту, раскладывая письма в папки по темам. Через какое-то время эта работа вам надоест, и вы захотите ее автоматизировать. Один из возможных подходов – проанализировать собственное поведение и выписать правила, которыми вы руководствуетесь при сортировке писем. Однако это громоздкое и отнюдь не совершенное решение. Некоторые правила вы упустите из виду, другие сформулируете излишне детально. Гораздо лучше выбрать какой-то набор метаданных о письмах, задать пары (тело письма, имя папки) и поручить алгоритму вывести наилучший набор правил. Такие пары называются обучающими данными, а получившийся набор правил можно будет применить к будущим письмам, которых алгоритм еще не видел. Это и есть машинное обучение в простейшем виде.

Представим, что нам принадлежит большая сеть ресторанов, и мы хотим открыть в неком городе новое заведение. Мы нашли несколько точек в городе, где есть возможность приобрести помещение и организовать там ресторан. Нам важно, чтобы через определенное время он стал приносить прибыль - точнее, хочется открыть его в той точке, в которой прибыль окажется максимальной. Поставим задачу: для каждого возможного размещения ресторана предсказать прибыль, которую он принесет в течение первого года. Множество всех возможных точек размещения называется *пространством объектов* и обозначается через X. Величина, которую мы хотим определять (т.е. прибыль ресторана), называется *целевой переменной*, а множество ее значений - пространством ответов Y. В нашем случае пространство ответов является множеством вещественных чисел: $Y = R$.

Мы не являемся специалистами в экономике, поэтому не можем сделать такие прогнозы на основе своих экспертных знаний. У нас есть лишь примеры - поскольку мы владеем целой сетью ресторанов, то имеем данные по достаточно большому числу ранее открытых ресторанов и по их прибыли в течение первого года. Каждый такой пример называется обучающим, а вся их совокупность - *обучающей выборкой*, которая обозначается как $X = \{(x_1, y_1), \ldots, (x_l, y_l)\}$, где $x_1, \ldots, x_l$ - обучающие объекты, а l - их количество. Особенность обучающих объектов состоит в том, что для них известны ответы $y_1, \ldots, y_l$.

Отметим, что объекты — это некие абстрактные сущности (точки размещения ресторанов), которыми компьютеры не умеют оперировать напрямую. Для дальнейшего анализа нам понадобится описать объекты с помощью некоторого набора характеристик, которые называются признаками (или факторами). Вектор всех признаков объекта x называется признаковым описанием этого объекта. Далее мы будем отождествлять объект и его признаковое описание. Признаки могут быть очень разными: бинарными, вещественными, категориальными (принимают значения из неупорядоченного множества), ординальными (принимают значения из упорядоченного множества), множествозначными (set-valued, значения являются подмножествами некоторого универсального множества). Признаки могут иметь сложную внутреннюю структуру: так, в качестве признака для конкретного человека в задаче предсказания его годового дохода может служить фотография. Разумеется, фотографию можно представить и как некоторое количество бинарных или вещественных признаков, каждый из которых кодирует соответствующие пиксель изображения. Однако, работа с изображением как с одной сложной структурой позволяет вычислять по нему различные фильтры, накладывать требование инвариантности ответа к сдвигам и т.д. Именно на работе со сложными данными специализируется активно развивающееся сейчас глубинное обучение (deep learning).

В задаче полезными могут оказаться признаки, связанные с демографией (средний возраст жителей ближайших кварталов, динамика изменения количества жителей) или недвижимостью (например, средняя стоимость квадратного метра в окрестности, количество школ, магазинов, заправок, торговых центров, банков поблизости). Разработка признаков (feature engineering) для любой задачи является одним из самых сложных и самых важных этапов анализа данных.

Описанная задача является примером задачи *обучения с учителем* (supervised learning), а более конкретно задачей *регрессии* - именно так называются задачи с вещественной целевой переменной. Перечислим несколько других видов задач обучения с учителем:

1. $Y = \{0, 1\}$ – бинарная классификация. Например, мы можем предсказывать, кликнет ли пользователь по рекламному объявлению, вернет ли клиент кредит в установленный срок, сдаст ли студент сессию, случится ли определенное заболевание с пациентом (на основе, скажем, его генома).

2. $Y = \{1, \ldots, K\}$ – многоклассовая (multi-class) классификация. Примером может служить определение предметной области для научной статьи (математика, биология, психология и т.д.).

3. $Y = \{0,1\}^K$ – многоклассовая классификация с пересекающимися классами (multi-label classification). Примером может служить задача автоматического проставления тегов для ресторанов (логично, что ресторан может одновременно иметь несколько тегов).

4. Частичное обучение (semi-supervised learning) – задача, в которой для одной части объектов обучающей выборки известны и признаки, и ответы, а для другой только признаки. Такие ситуации возникают, например, в медицинских задачах, где получение ответа является крайне сложным (например, требует проведения дорогостоящего анализа).

Существует также обучение без учителя– класс задач, где ответы неизвестны или вообще не существуют, и требуется найти некоторые закономерности в данных лишь на основе признаковых описаний:

1. Кластеризация – задача разделения объектов на группы, обладающие некоторыми свойствами. Примером может служить кластеризация документов из электронной библиотеки или кластеризация абонентов мобильного оператора.

2. Оценивание плотности – задача приближения распределения объектов. Примером может служить задача обнаружения аномалий, в которой на этапе обучения известны лишь примеры «правильного» поведения оборудования (или, скажем, игроков на бирже), а в дальнейшем требуется обнаруживать случаи некорректной работы (соответственно, незаконного поведения игроков). В таких задачах сначала оценивается распределение «правильных» объектов, а затем аномальными объявляются все объекты, которых в рамках этого распределения получают слишком низкую вероятность.

3. Понижение размерности – задача генерации таких новых признаков, что их меньше, чем исходных, но при этом с их помощью задача решается не хуже (или с небольшими потерями качества, или лучше – зависит от постановки). К этой же категории относится задача построения латентных моделей, где требуется описать процесс генерации данных с помощью некоторого (как правило, небольшого) набора скрытых переменных. Примерами являются задачи тематического моделирования и построения рекомендаций.

Вернемся к нашей задаче по предсказанию прибыли ресторана. Предположим, что мы собрали обучающую выборку и изобрели некоторое количество признаков. Результатом будет матрица «объекты-признаки» $X \in R^{l \times d}$ (l – число объектов, d – число признаков), в которой каждая строка содержит признаковое описание одного из обучающих объектов. Таким образом, строки в этой матрице соответствуют объектам, а столбцы – признакам.

Нашей задачей является построение функции $a : X \to Y$, которая для любого объекта будет предсказывать ответ. Такая функция называется *алгоритмом или моделью* (hypothesis). Понятно, что нам подойдет далеко не каждый алгоритм – например, вряд ли

мы извлечем какую-то выгоду из алгоритма $a(x) = 0$, который предсказывает нулевую прибыль для любого ресторана независимо от его признаков. Чтобы формализовать соответствие алгоритма нашим ожиданиям, нужно ввести функционал качества, измеряющий качество работы алгоритма. Отметим, что если функционал устроен так, что его следует минимизировать, то логичнее называть его функционалом ошибки. Крайне популярным функционалом в задаче регрессии является среднеквадратичная ошибка (mean squared error, MSE):

$$Q(a, X) = \frac{1}{l}\sum_{i=1}^{l}(a(x_i) - y_i)^2$$

Чем более маленькое значение этого функционала дает алгоритм, тем он лучше. Данный функционал является очень удобным благодаря дифференцируемости и простоте, но, как мы увидим дальше в курсе, обладает и большим количеством недостатков. В большинстве случаев функционалы представляют собой сумму ошибок на отдельных объектах обучающей выборки. Функция, измеряющая ошибку одного предсказания, называется функцией потерь $L : Y \times Y \rightarrow R_+$ (в нашем случае $L(y, z) = (y - z)^2$).

Заметим, что именно функционал качества будет определять во всех дальнейших рассуждениях, какой алгоритм является лучшим. Если метрика выбрана неудачно и не соответствует бизнес-требованиям или особенностям данных, то все дальнейшие действия обречены на провал. Именно поэтому выбор базовой метрики является крайне важным этапом в решении любой задачи анализа данных. Она не обязательно должна обладать хорошими математическими свойствами (непрерывность, выпуклость, дифференцируемость и т.д.), но обязана отражать все важные требования к решению задачи.

Как только функционал качества зафиксирован, можно приступать к построению алгоритма *a(x)*. Как правило, для этого фиксируют некоторое *семейство алгоритмов* A, и пытаются выбрать из него алгоритм, наилучший с точки зрения функционала. В машинном обучении было изобретено большое количество семейств алгоритмов, и, наверное, самым простым и самым тщательно изученным среди них является семейство *линейных моделей*, которые дают предсказание, равное линейной комбинации признаков:

$$A = \{a(x) = \omega_0 + \omega_1 x_1 + \cdots + \omega_d x_d | \omega_0, \omega_1, \ldots, \omega_d \in R\},$$

где через x_i обозначается значение i-го признака у объекта x. Лучшая из таких моделей будет выбираться путем минимизации MSE-функционала:

$$\frac{1}{l}\sum_{i=1}^{l}\left(\omega_0 + \sum_{j=1}^{d}\omega_j x_{ij} - y_i\right)^2 \to \min_{\omega_0,\omega_1,\ldots,\omega_d}.$$

Через x_{ij}здесь обозначается значение $j-$го признака на $i-$м объекте. Процесс поиска оптимального алгоритма называется *обучением*.

Зачастую возникает потребность в *предобработке данных* до начала построения модели. Здесь может идти речь о некотором ряде манипуляций:

• Некоторые модели хорошо работают только при выполнении определенных требований. Так, для линейных моделей крайне важно, чтобы признаки были нормированными, то есть измерялись в одной шкале. Примером способа нормировки данных является вычитание среднего и деление на дисперсию каждого столбца в матрице «объекты-признаки».

• Бывает, что в выборку попадают *выбросы* – объекты, которые не являются корректными примерами из-за неправильно посчитанных признаков, ошибки сбора данных или чего-то еще. Их наличие может сильно испортить модель.

• Некоторые признаки могут оказаться шумовыми, то есть не имеющими никакого отношения к целевой переменной и к решаемой задаче. Примером, скорее всего, может служить признак «фаза луны в день первого экзамена» в задаче предсказания успешности прохождения сессии студентом.

Как показывает практика, простейшая предобработка данных может радикально улучшить качество итоговой модели. Во время обучения модели очень важно следить за тем, чтобы не произошло *переобучения* (overfitting). Разберем это явление на примере. Допустим, что мы выбрали очень богатое семейство алгоритмов, состоящее из всех возможных функций:

$$A = \{a : X \to Y\}.$$

В этом семействе всегда будет алгоритм, не допускающий ни одной ошибки на обучающей выборке, который просто запоминает ее:

$$a(x) = \begin{cases} y_{i,} & \text{если } x = x_i \\ 0, & \text{если } x \notin X \end{cases}$$

Очевидно, что такой алгоритм нас не устраивает, поскольку для любого нового ресторана предскажет нулевую прибыль. Алгоритм оказался переобученным – он слишком сильно подогнался под обучающую выборку, не выявив никаких закономерностей в ней. Существует ряд методов, направленных на борьбу с переобучением, которые мы будем

обсуждать на следующих занятиях. Впрочем, одну из идей мы можем обсудить уже сейчас. В нашем примере переобучение возникло из-за большой сложности семейства – алгоритмом могла оказаться любая функция. Очевидно, что если бы мы ограничили себя только линейными моделями, то итоговый алгоритм уже не смог бы запомнить всю выборку. Таким образом, можно бороться с переобучением путем контроля сложности семейства алгоритмов – чем меньше у нас данных для обучения, тем более простые семейства следует выбирать.

После того, как модель построена, нам нужно оценить, насколько хорошо она будет работать на новых данных. Для этого, например, можно в самом начала отложить часть обучающих объектов и не использовать их при построении модели. Тогда можно будет измерить качество готовой модели на этой отложенной выборке, получив тем самым оценку того, насколько она готова к работе на новых данных.

Основные этапы решения задачи анализа данных:

1. Постановка задачи;
2. Отбор признаков;
3. Формирование выборки;
4. Выбор метрики качества;
5. Предобработка данных;
6. Построение модели;
7. Оценивание качества модели.

Классический метод обучения, называемый минимизацией эмпирического риска (empirical risk minimization, ERM), заключается в том, чтобы найти для заданной модели A алгоритм a, доставляющий минимальное значение функционалу качества Q на заданной обучающей выборке X^l: $\mu(X^l) = arg \min\limits_{a \in A} Q(a, X^l)$.

Функционал среднего риска

Рассмотрим произвольный алгоритм $a: X \to Y$. Он разбивает множество X на непересекающиеся области $A_y = \{x \in X \mid a(x) = y\}, y \in Y$. Вероятность того, что появится объект класса *y* и алгоритм *a* отнесёт его к классу *s*, равна $P_y \mathrm{P}(A_S|y)$. Каждой паре $(y, s) \in Y \times Y$ поставим в соответствие величину потери λ_{ys} при отнесении объекта класса *y* к классу *s*. Обычно полагают $\lambda_{yy} = 0$, и $\lambda_{ys} > 0$ при $y \neq s$. Соотношения потерь на разных классах, как правило, известны заранее.

Функционалом среднего риска называется ожидаемая величина потери при классификации объектов алгоритмом a:

$$R(a) = \sum_{y \in Y} \sum_{s \in Y} \lambda_{ys} P_y \mathrm{P}(A_s | y).$$

Если величина потерь одинакова для ошибок любого рода, $\lambda_{ys} = [y \neq s]$, то средний риск *R(a)* совпадает с вероятностью ошибки алгоритма a.

Основные инструменты машинного обучения

Представлены некоторый список из наиболее популярных инструментов и технологий, используемых для реализации и использования машинного обучения на основных языках программирования:

Машинное обучение в JavaScript.

Разработчики JavaScript используют различные программные платформы для обучения и развертывания моделей машинного обучения в браузере.

1. TensorFlow.js — это библиотека с открытым исходным кодом, позволяющая полностью запускать программы машинного обучения в браузере. Это наследник Deeplearn.js, который больше не поддерживается. С помощью библиотеки можно использовать универсальные и интуитивно понятные API-интерфейсы для определения, обучения и развертывания моделей с нуля прямо в браузере. Кроме того, он автоматически предлагает поддержку WebGL и Node.js.

2. Machine learning tools - Библиотека инструментов машинного обучения представляет собой сборник ресурсоемких инструментов с открытым исходным кодом для поддержки широко распространенных функций машинного обучения в браузере. Эти инструменты обеспечивают поддержку нескольких алгоритмов машинного обучения, включая обучение без учителя, обработку данных, искусственные нейронные сети (ANN), математику и регрессию. Если вы начинаете изучать Python и ищете что-то похожее на Scikit-learn для машинного обучения в браузере JavaScript, этот набор инструментов может помочь вам.

3. Keras.js — еще одна популярная платформа с открытым исходным кодом, которая позволяет запускать модели машинного обучения в браузере. Она предлагает поддержку режима GPU с использованием WebGL. Если есть модели в Node.js, то будет запускать их только в режиме процессора. Keras.js также предлагает поддержку моделей, обученных с использованием любой серверной среды, такой как Microsoft Cognitive Toolkit (CNTK). Некоторые из моделей Keras, которые могут быть развернуты в браузере на

стороне клиента, включают Inception v3 (обучение по ImageNet), 50-слойную сеть (обучение по ImageNet) и обучение по MNIST.

4. STDLib — это библиотека с открытым исходным кодом для поддержки приложений JavaScript и Node.js. Если необходима библиотека, которая подчеркивает поддержку в браузере для научных и числовых веб-приложений машинного обучения, STDLib может удовлетворить ваши потребности. Библиотека поставляется с обширными и продвинутыми математическими и статистическими функциями, которые помогут в создании высокопроизводительных моделей машинного обучения. А также можете использовать его расширенные утилиты для создания приложений и других библиотек. Кроме того, если нужна среда для визуализации и анализа данных, то найдете STDLib стоящей.

Машинное обучение в Python:

1. NumPy — это очень популярная библиотека на python для обработки больших многомерных массивов и матриц с помощью большого набора математических функций высокого уровня. Это очень полезно для фундаментальных научных вычислений в машинном обучении. Это особенно полезно для линейной алгебры, преобразования Фурье и случайных чисел. Высококачественные библиотеки, такие как TensorFlow, используют NumPy для манипулирования Tensors.

2. Keras — очень популярная библиотека машинного обучения для Python. Это высокоуровневый API нейронных сетей, способный работать поверх TensorFlow, CNTK или Theano. Он может работать как на процессоре, так и на GPU. Keras — это действительно для начинающих data scientist, которые хотят строить и проектировать нейронную сеть. Одним из лучших преимуществ Keras является то, что он позволяет легко и быстро создавать прототипы.

Машинное обучение в C#:

1. ML.NET — это кроссплатформенная среда машинного обучения с открытым исходным кодом, которая делает машинное обучение доступным для разработчиков .NET. ML.NET позволяет .NET разработчикам разрабатывать свои собственные модели и внедрять индивидуальное машинное обучение в свои приложения, используя .NET, даже без предварительного опыта разработки или настройки моделей машинного обучения. Наряду с этими возможностями ML в нем также представлен первый проект API-интерфейсов .NET для моделей обучения, использующих модели для предсказаний, а также

основные компоненты этой среды, такие как алгоритмы обучения, преобразования и данные ML структур.

2. Microsoft Azure ML Studio - сервис использует облачные хранилища для машинного обучения. Microsoft предоставляет как платную, так и бесплатную версии, возможность пользоваться готовыми алгоритмами (своими и созданными сторонними компаниями). Тестировать платформу можно через анонимный профиль, а модели превращать в API и предоставлять другим сервисам. В бесплатной версии пользователям доступно 10 Гб, чтобы сохранять данные.