Глава 9

Машинное обучение

Рассматриваемые в этой главе методы решающих деревьев (см. раздел 9.1), случайного леса (см. раздел 9.2.2), изолирующего леса (см. раздел 9.2.3), нейронных сетей (см. раздел 9.5) — это методы, которые в настоящее время традиционно объединяют под названием методов *машинного обучения*[1]. Часто говорят, что машинное обучение решает задачу об аппроксимации. Например, пусть есть некоторый набор данных, состоящий из N векторов $\mathbf{x}_i \in \mathrm{R}^n$, называемых векторами *признаков*[2], и соответствующие им N значений y_i. Подразумевается, что y_i получен из $\mathbf{x}_i$ с помощью некоторого соотношения. Конкретный вид этого соотношения чаще всего неизвестен, либо вычислительно непрактичен, кроме того, часто предполагается, что сама эта зависимость существует в статистическом смысле, т.е. $\mathbf{x}_i$ и y_i могут быть отягощены некоторой неустранимой погрешностью. Мы бы хотели получить некоторую функцию $f(\mathbf{x})$, наилучшим образом аппроксимирующую неизвестную закономерность. Если y_i являются действительными числами, то говорят о *задаче регрессии*. Если y_i могут принимать дискретные значения, то говорят о *задаче классификации*, а сами y_i называют *метками*[3] классов.

Приведем другой пример. Пусть есть выборка из N реализаций случайного вектора $\mathbf{x}_i$, мы бы хотели получить такую функцию $f(\mathbf{x})$, которая аппроксимирует многомерную функцию плотности вероятности рассматриваемой случайной величины $p(\mathbf{x})$.

Или, например, пусть есть выборка из N реализаций случайного вектора $\mathbf{x}_i$, и пусть есть некоторая случайная величина $\mathbf{z}$ с нормальным распределением, нулевым средним и единичной дисперсией. Мы бы хотели получить такую функцию $f(\mathbf{z})$, чтобы случайный вектор $\xi = f(\mathbf{z})$ оказался бы распределен таким же образом как и случайный вектор $\mathbf{x}$.

Заметим, что с задачей регрессии мы сталкивались в разделах 3 и 5.1.1. Задача оценки плотности вероятности по выборке реализаций случайного

[1] *machine learning*
[2] *features*
[3] *labels*

вектора решалась в главе 6. Во всех случаях сначала мы постулировали некоторый параметрический вид описываемой зависимости (например, линейная зависимость в разделе 3, или нормальное распределение в главе 6), а затем задача сводилась к оцениванию неизвестных параметров. Таким образом, из семейства функций выбиралась одна конкретная.

Понятно, что при работе с эмпирическими моделями чем шире класс функций из которых мы выбираем, тем лучше. Существует ряд классических методов, позволяющих параметризовать широкий класс функций, к таковым можно отнести ряды Фурье, всевозможные сплайны, интерполяционные полиномы, включая классические ортогональные полиномы, и т.п. Пожалуй, общее свойство всех этих методов состоит в том, что описываемое семейство функций допускает запись в виде формулы, пригодную для последующего практического использования. Машинное обучение представляет нам дополнительный способ алгоритмического задания аппроксимирующей функции. Нас удовлетворит, если для любого $\mathbf{x}$ можно будет получить соответствующее числовое значение $f(\mathbf{x})$.

Термин *обучение* используется, чтобы подчеркнуть, что искомый алгоритм не задается жестко пользователем, а подбирается вычислительной машиной на основе известных данных. В простом случае, как например для метода нейронных сетей, это означает подбор неизвестных весовых коэффициентов. Для методов основанных на деревьях, мы подразумеваем процесс построения структур данных (деревьев), обладающих требуемыми свойствами.

Стоит уточнить, что подобный алгоритм не просто должен уметь вычислять выходное значение, но и делать это в некотором смысле быстро. Так, например, *метод k ближайших соседей* основан на свойстве k-мерных деревьев (см. раздел 2.4.3) эффективно находить для любой точки $\mathbf{x}$ ближайшие точки из заданного набора X. Действительно, имея выборку $\mathbf{x}_i$ и соответствующие им y_i, для решения задачи регрессии мы можем организовать словарь на основе k-мерного дерева. А в качестве предсказания (т.е. вычисления $f(\mathbf{x})$) мы выполним поиск k записей $\mathbf{x}_i$, ближайших к точке $\mathbf{x}$ в эвклидовой метрике, и усредним соответствующие этим записям y_i.

Кроме вычислительных затрат, важной характеристикой моделей, построенных с помощью методов машинного обучения, является их качество, т.е. способность адекватно предсказывать требуемые величины. Количественные меры качества моделей называются *метриками качества* и рассматриваются далее в разделе 9.3.

Отметим место таких методов в физических науках. Можно отметить два самых популярных применения методов машинного обучения. Во-первых, построение эмпирических моделей, например таких как калибровочные зависимости. Напомним, часто в физике бывает так, что построить строгую модель не удается — это оказывается либо сложно технически, либо не все влияющие факторы измеряются в эксперименте. Здесь в контексте машинного обучения и задачи классификации интересно упомянуть и о задаче фильтрации измерений.

Во-вторых, ускорение вычислений. Часто бывает так, что строгие фи-

зические модели достаточно затратны с вычислительной точки зрения, и, хотелось бы выполнить расчет на некоторой сетке параметров, а затем интерполировать результаты для значений находящихся между узлов сетки. В этом случае методы машинного обучения могут быть хорошей альтернативой классическим методом интерполяции, особенно если речь идет о пространствах большой размерности.

9.1 Решающее дерево

Решающее дерево[4], или *дерево принятия решений* — один из методов построения моделей машинного обучения для задач регрессии и классификации. Чаще всего решающее дерево используется как компонент для построения методов ансамблей, таких как, например, случайный лес (см. раздел 9.2.2).

В основе методов машинного обучения, основанных на деревьях, лежит идея о том, что кусочно-постоянная функция в многомерном пространстве может быть эффективно (с вычислительной точки зрения) представлена с помощью бинарного дерева. В одномерном случае, одним из наивных способов численного представления кусочно-постоянной функции является составление отсортированной таблицы пар x_i, y_i с последующим поиском x_i ближайшего слева к произвольному x. В многомерном случае такой подход не срабатывает из-за необходимости иметь число записей, показательно зависящее от размерности пространства n.

В разделе 2.4.3 обсуждалась организация коллекции из N многомерных векторов $\mathbf{x}_i \in \mathrm{R}^n$ в k-мерное дерево, структуру данных, позволяющую эффективно (асимптотическая вычислительная сложность $O(\log N)$) искать элементы коллекции и ближайших соседей. Посмотрим на такое дерево под несколько другим углом зрения. Занумеруем листья дерева, и для каждого $\mathbf{x} \in \mathrm{R}^n$ будем пытаться найти вхождение этого элемента в исходной коллекции: неудачный поиск всегда будет заканчиваться в том или ином листе по построению алгоритма. Попытаемся графически представить диаграмму неудачного поиска: т.е. покрасим каждую точку R^n в индивидуальный цвет, соответствующий номеру листа. Очевидно, что подобная диаграмма будет состоять из не пересекающихся многомерных параллелепипедов, замощающих всё пространство R^n. Причем их стороны строго параллельны координатным осям рассматриваемого пространства, а некоторые параллелепипеды могут иметь стороны, расположенные на бесконечности. Достаточно приписать теперь каждому листу свое значение $\hat{y}_i$ (которое в нем и будет храниться), чтобы получить многомерную кусочно-постоянную функцию.

Итак, многомерное двоичное дерево может быть использовано для эффективного вычисления кусочно-постоянной функции. Причем значения этой функции определяются параметрами $\hat{y}_i$, заданными в листьях дерева, а границы прямоугольников определяются значениями $\hat{x}_i$, k_i, заданными в

[4] *decision tree*

узлах дерева. Обратим внимание, что в отличии от задачи хранения и поиска заданной коллекции многомерных векторов, величины $\hat{y}_i$ в каждом листе и $\hat{x}_i$, k_i в каждом промежуточном узле являются параметрами. Кроме того, параметром является число листьев дерева, так как каждый отдельный интервал кусочно-постоянной функции представляется отдельным листом.

Возникает вопрос, как эффективно с точки зрения вычислительной сложности и оптимально с точки зрения аппроксимации некоторой функции подобрать неизвестные параметры $\hat{y}_i$, $\hat{x}_i$, и k_i, чтобы полностью задать дерево, которое бы представляло искомую функцию, например в задаче классификации или регрессии? Понятно, что в разделе 2.4.3 мы стремились построить дерево, имеющее по возможности наименьшую глубину, что положительно сказывалось на эффективности поиска по коллекции. Теперь мы смещаем внимание на эффективность аппроксимации неизвестной зависимости кусочно-постоянной функцией, численно представляемой многомерным бинарным деревом.

При этом есть основания полагать, что вычислительная сложность не пострадает. Во-первых, напомним, что даже случайное бинарное дерево имеет среднюю глубину пропорциональную логарифму числа узлов $O(\log N)$ (см. раздел 2.4.3). Во-вторых, частой практикой является ограничение глубины дерева, задаваемое перед построением дерева. В терминах кусочно-постоянной функции это эквивалентно ограничению числа индивидуальных сегментов функции.

9.1.1 Задача классификации

Для того, чтобы предложить конкретный метод построения бинарного дерева принятия решений, начнем с задачи бинарной классификации, пусть $y_i \in \{0, 1\}$. Понятно, что с точки зрения вычислительной эффективности было бы удобно использовать *жадный*[5] способ построения дерева, т.е. задать одно универсальное правило разбиения данных на два подмножество и применять его рекуррентно. Напомним, что жадным алгоритмом называется алгоритм, который принимает последовательность локально-оптимальных решений с целью получить глобально-оптимальное решение. В нашем случае, строго гарантировать глобальную оптимальность не удаётся, но оказывается, что на практике следующий метод работает эффективно.

Если представить, что глубина дерева ограничена 1, то всё дерево будет состоять из одного корня и двух листьев, тогда параметры корня дерева $\hat{x}_0, k_0$ определят границу раздела пространства на две части, в каждой из которых функция заданная деревом будет принимать некоторое постоянное значение. С точки зрения здравого смысла понятно, что нам бы хотелось провести границу так, чтобы по обе стороны от неё содержащиеся там точки в большинстве своем принадлежали какому-то из двух классов. В

[5] *greedy*

таком случае, количество ошибок неправильной классификации, совершаемых при использовании аппроксимации, было бы минимальным.

Одной из возможных количественных мер оценки чистоты выборки является *примесь Джини*[6]:

$$G \equiv p_0(1-p_0) + p_1(1-p_1) = 2p_0(1-p_0), \tag{9.1}$$

где

$$p_0 \equiv \frac{N_0}{N_0+N_1}, \tag{9.2}$$

$$p_1 \equiv \frac{N_1}{N_0+N_1}. \tag{9.3}$$

Здесь N_0 и N_1 обозначают количество точек принадлежащих нулевому и первому классам соответственно. Критерий легко обобщается на случай многомерной классификации:

$$G \equiv \sum_{j=0}^{K-1} p_j(1-p_j), \tag{9.4}$$

$$p_j \equiv \frac{N_j}{\sum_{k=0}^{K-1} N_k}. \tag{9.5}$$

где K — полное число различных классов.

Примесь Джини имеет смысл аналогичный дисперсии для непрерывных величин. Это легко уяснить из следующего вспомогательного примера. Пусть задан случайный вектор β, реализации которого могут принимать лишь два значения: $(1,0)$ с вероятностью p_0 и $(0,1)$ с вероятностью $p_1 = 1-p_0$. С помощью нехитрых вычислений можно убедиться, что среднее такого случайного вектора (p_0, p_1), а матрица ковариации:

$$\operatorname{cov}\beta = \begin{pmatrix} p_0(1-p_0) & -p_0p_1 \\ -p_1p_0 & p_1(1-p_1) \end{pmatrix}. \tag{9.6}$$

Идея состоит в том, чтобы сосчитать взвешенную сумму примесей Джини G для обоих частей (подвыборок) разбиения пространства признаков, а затем подобрать разбиение таким образом, чтобы рассматривая сумма была минимально возможной. Для этого обозначим N_l — количество данных в полупространстве соответствующем левому поддереву, N_r — правому поддереву, G_l — примесь Джини для левого поддерева, G_r — для правого, $\theta \equiv (x_0, k_0)$ параметры разбиения. Понятно, что все величины зависят от параметров разбиения θ, но для краткости не будем обозначать это на записи. Тогда можно подобрать параметры разбиения таким образом, чтобы относительная взвешенная примесь Джини оказалась минимальной:

$$\theta = \arg\min_{\theta} \left(\frac{N_l}{N_l+N_r} G_l + \frac{N_r}{N_l+N_r} G_r \right). \tag{9.7}$$

[6] *Giny impurity*

Далее, аналогичным образом индивидуально разобьем левое и правые поддеревья. Окончательно, каждый лист будет накрывать область пространства признаков, содержащую хотя бы одну точку данных, либо, если при построении задано ограничение по глубине дерева, несколько точек. Окончательно, каждому листу $\hat{y}_i$ припишем класс, который, например, наиболее часто встречается в этой области.

9.1.2 Задача регрессии

Если мы решаем задачу регрессии, тогда нам, очевидно, нужна другая функция для поиска оптимального разделения. Наиболее очевидный вариант — подсчитать выборочную дисперсию подвыборки:

$$V \equiv \frac{1}{N-1} \sum_{i=1}^{N} (y_i - \bar{y})^2, \tag{9.8}$$

$$\bar{y} \equiv \frac{1}{N} \sum_{i=1}^{N} y_i. \tag{9.9}$$

Чем меньше выборочная дисперсия, тем ближе друг к другу значения y_i в подвыборке. В случае нулевой дисперсии все значения равны друг другу, значит такое разбиение было бы для нас идеальным. Следовательно, можно определить критерий выбора параметров разбиения следующим образом:

$$\theta = \arg\min_{\theta} \left(\frac{N_l}{N_l + N_r} V_l + \frac{N_r}{N_l + N_r} V_r \right) = \\ = \arg\max_{\theta} \frac{N_l N_r}{(N_l + N_r)^2} (\bar{y}_l - \bar{y}_r)^2, \tag{9.10}$$

где как и ранее N_l — количество данных в полупространстве соответствующем левому поддереву, N_r — правому поддереву, V_l, V_r — дисперсия для левого и правого поддеревьев, соответственно, y_l, y_r — выборочное среднее для левого и правого поддеревьев.

Интересно, что в области компьютерного зрения критерий (9.10) был предложен японским исследователем Нобуюки Оцу (см. Otsu 1979), метод Оцу позволяет динамически подбирать порог для выделения ярких пикселей на тёмном фоне. В отличии, например, от модели смеси двух нормальных распределений (см. раздел 6.1), который тоже можно было бы применить для определения оптимального порога, в этом методе не делается предположений о распределении значений тёмных (фоновых) пикселей, и распределении значений светлых пикселей (пикселей объекта).

9.2 Методы на основе ансамблей деревьев

Даже не применяя формальные количественные меры эффективности, которые рассматриваются позже в разделе 9.3, очевидно следующее поведение

решающих деревьев. Неограниченное в глубину решающее дерево разбивает пространство признаков на маленькие области, игнорирующие присутствие погрешности во входных данных, и, таким образом, предсказательная мощь модели снижается за счет того, что шум не усредняется по соседним областям пространства. Этот эффект называется *переобучением*. При ограничении глубины дерева, напротив, пространство разбивается на несколько больших по размеру областей, таким образом аппроксимация получается слишком грубой и не учитывающей отдельные особенности.

Одним из элегантных способов преодоления проблемы являются методы, основанные на ансамблях деревьев, ансамбль (т.е. набор) деревьев логично называют *лесом*. До настоящего момента алгоритм построения решающего дерева был полностью детерминирован и зависел лишь от входных данных. Однако, мы понимаем, что вектора признаков $\mathbf{x}_i$ сами по себе могут быть подвержены, например, погрешностям измерений, т.е. при повторном проведении идентичного эксперимента значения векторов будут отличаться в рамках точности измерений. А это повлечёт за собой изменение структуры дерева, и, как следствие, мы получим немного другую аппроксимирующую функцию.

Однако, другой реализации набора данных у нас нет, поэтому попробуем сымитировать выборочный шум внося элемент случайности в построение дерева. При каждом новом выполнении процедуры будет получаться новая реализация дерева, т.е. выборка реализаций деревьев некоторого размера, или ансамбль. Очевидно, что для одного и того же $\mathbf{x}$ каждое дерево может теперь возвращать различный результат. Можно условиться, что для задачи регрессии финальным ответом нашей модели будет являться усредненный ответ всех деревьев, а для задачи классификации, например, будет устраиваться голосование — итоговым классом будет назначен тот, за который высказалось большинство деревьев.

9.2.1 Bootstrap aggregation

Один из способов внести случайность является так называемый *bootstrap aggregation*[7], или сокращённо *bagging*. Этот метод впервые был предложен в работе американского исследователя Лео Бреймана (см. Breiman 1996). Для построения ансамбля деревьев используется *бутстрап*[8], общий подход, который имитирует выборочный шум с помощью построения подвыборок с возвратом. Для выборки из N векторов $\mathbf{x}_i$ необходимо задать размер подвыборки M, а затем случайно выбрать M элементов из исходной выборки. Вероятность выбора каждого элемента составляет $1/N$ и не зависит от соответствующего значения $\mathbf{x}_i$. Поэтому можно говорить, что исходная выборка и все реализации случайной подвыборки имеют одинаковые статистические свойства, однако различную реализацию выборочного шума.

[7] К сожалению, общеупотребимый перевод термина на русский язык пока не успел сформироваться.

[8] *bootstrap*

Для построения каждого дерева ансамбля берётся уникальное случайное подмножество исходных данных. С одной стороны, поскольку мы считаем, что случайные подвыборки меньшего размера статистически похожи на исходный набор данных, мы ожидаем, что все деревья смогут решить исходную задачу. С другой стороны, теперь каждое из решающих деревьев получится немного отличным, так как каждое из деревьев строится на своём случайном подмножестве данных. На плоскости это будет графически соответствовать тому, что прямоугольники расположены немного в разных местах и имеют различную форму. За счёт последующего усреднения по ансамблю деревьев, результирующая модельная функция будет иметь уже гораздо больше отдельных участков постоянного значения, что позволяет более плавно описывать изменения аппроксимируемой зависимости, чем это удалось бы сделать с помощью отдельного дерева.

9.2.2 Случайный лес

Метод *случайного леса*[9] развивает подход bootstrap aggregation. Деревья, построенные по методу bootstrap aggregation оказываются скореллированными друг с другом. Так, например, разбиения на верхних уровнях почти всегда используют одни и те же признаки, хотя и выбирают немного разные пороги. Оказывается, что можно улучшить эффективность метода, внеся дополнительное разнообразие в получаемые модели (см. Breiman 2001). Так, предлагается на каждом шаге выбирать подмножество признаков по которым разбиение доступно, а остальные признаки на данном шаге игнорируются. Т.е. вместо того, чтобы выполнять поиск минимума целевой функции разбиения по всем координатам вектора $\mathbf{x}$, выбирается некоторое случайное подмножество координат, а далее происходит процедура поиска наилучшего разбиения.

9.2.3 Ансамбль случайных деревьев

Известны и другие способы ещё большей рандомизации деревьев. Например, *чрезвычайно случайные деревья*[10] строятся по следующему принципу (см. Geurts, Ernst и Wehenkel 2006). Как и в методе случайного леса для каждого разбиения выбирается некоторое случайное подмножество рассматриваемых признаков, но затем, в отличии от метода случайного леса, порог разбиения для каждого признака назначается случайным образом, необходимо лишь гарантировать, чтобы в каждое из потенциальных поддеревьев попала хотя бы одна точка данных. Затем, среди этих разбиений-кандидатов выбирается оптимальный в смысле оптимизации (9.7) или (9.10). При использовании модели выполняется усреднение по ансамблю как и в предыдущих случаях. Понятно, что количество таких кандидатов является некоторым настроечным параметром. На первый взгляд удивительно, но оказывается, что даже такой ансамбль полностью случайных деревьев (т.е.

[9] *random forest*
[10] *extremely randomized tree* или *extra-tree*

когда для каждого разделения выбирается один случайный признак и один случайный порог) способен эффективно решать задачи классификации и регрессии. Очевидным преимуществом такого подхода перед методом случайного леса является скорость построения деревьев.

Изолирующий лес

Чрезвычайно случайные деревья, в которых на каждом этапе разбиение производится полностью случайным образом, практически независимо от значений входных данных, имеют ещё одного интересное применение — поиск вылетающих значений выборки или аппроксимация плотностей вероятности выборки. Вылетающими значениями обычно называют такие элементов выборки, которые в пространстве параметров отстоят от соседей дальше, чем это принято в среднем, т.е. населяют области с низкой плотностью вероятности. Понятно, что если мы умеем оценивать величину пропорциональную плотности вероятности случайной величины, то для каждой конкретной реализации можем получить значение этой плотности и сопоставить его со средним. В отличии от предыдущих задач, такую задачу называют задачей машинного обучения *без учителя*[11]. Отсутствие учителя означает отсутствие меток или набора y_i. Действительно, получить оценку плотности вероятности $\mathbf{x}_i$ можно используя лишь набор $\mathbf{x}_i$.

В одномерном случае пожалуй самый простой способ получить оценку плотности вероятности состоит в построении гистограммы. Однако, в случае пространств с большой размерностью число требуемых бинов многомерной гистограммы растёт степенным образом в зависимости от размерности пространства, что делает процедуру строительства многомерной гистограммы неэффективной как с вычислительной точки зрения, так и с точки зрения статистики: так как бинов будет неизбежно много больше, чем данных, то в большинство бинов данные не попадут вообще.

Метод изолирующего леса (см. Liu, Ting и Zhou 2008) представляет собой эффективный способ разбиения многомерного пространства с последующей оценкой плотности числа точек данных в каждой точке пространства $\mathbf{x}$. Изолирующий лес строится из чрезвычайно случайных деревьев, причём каждое разбиение выбирается случайным образом. Понятно, что раз метки y_i больше не участвуют в постановке задачи, то оценить оптимальность разбиения по формулам (9.7) или (9.10) больше нельзя. Следовательно, имеет смысл генерировать только один случайный кандидат в разбиения.

Построенные таким образом деревья обладают двумя полезными свойствами для решения сходной задачи. Во-первых, средняя глубина любого поддерева пропорциональна числу листьев (т.е. точек исходной выборки) содержащихся в этом поддереве. А именно, рассмотрим среднюю глубину

[11] *unsupervised*

d одномерного случайного поддерева с N листьями:

$$d(N) = \frac{1}{N-1}\sum_{k}^{N-1}\left(\frac{k}{N}(d(k)+1) + \frac{N-k}{N}(d(N-k)+1)\right) = \\ = 1 + \frac{1}{N-1}\sum_{k}^{N-1}\left(\frac{k}{N}d(k) + \frac{N-k}{N}d(N-k)\right) = \\ = 1 + \frac{2}{N(N-1)}\sum_{k}^{N-1} kd(k). \quad (9.11)$$

Если принять $d(1) = 0$ и $d(2) = 1$, тогда решением этого разностного уравнения является

$$d(N) = \sum_{k=2}^{N}\frac{2}{N} = 2\left(H(N) - 1\right), \quad (9.12)$$

где гармоническое число $H(N) \approx \gamma + \log N$, где $\gamma \approx 0.5772$ называется постоянной Эйлера.

Во-вторых, средний объём области пространства признаков, приписанный каждому узлу, оказывается пропорционален $2^{-d(N)}$. В отличии от сбалансированных бинарных деревьев, где все листья имеют почти одинаковую глубину, в случае со случайными деревьями листья могут иметь разную глубину. Поэтому глубокие листья соответствуют области пространства признаков с высокой плотностью точек, а листья с малой глубиной — области пространства с низкой плотностью точек.

Понять это можно и альтернативным способом. Допустим, что все значения $\mathbf{x}_i$ сосредоточены в некоторой области, и есть одно значение $\mathbf{x}_k$, отстоящее на расстоянии много большим размера этой области. Мы хотим построить разбиение этого набора на две ветви поддерева, выбрав случайным образом некоторый порог $\hat{x}_0$. Так как расстояние между вылетающим значением и кластером значений много больше размера кластера, то вероятность выбрать порог между ними приближается к 1, иными словами, вылетающая точка будет изолирована в своём персональном поддереве (левом или правом) с большой вероятностью. Понятно, что поддерево содержащее одну точку невозможно разбить дальше и оно образует лист.

Когда мы строим ансамбль случайных деревьев, то для каждой точки пространства $\mathbf{x}$ лист, накрывающий эту точку в каждом дереве, будет очевидно иметь немного разную глубину. Усредним эту глубину по ансамблю:

$$\bar{d}(\mathbf{x}) \equiv \frac{1}{M}\sum_{m=0}^{M} d_m(\mathbf{x}). \quad (9.13)$$

На практике удобно ввести нормированную величину в следующем виде:

$$s(\mathbf{x}) = -2^{-\frac{\bar{d}(\mathbf{x})}{d(N)}}, \quad (9.14)$$

где $d(N)$ вычисляется по формуле (9.12). Следовательно, величина $s(\mathbf{x}) \to -1$ для $\bar{d}(\mathbf{x}) \to 0$, и наоборот $s(\mathbf{x}) \to +1$ для $\bar{d}(\mathbf{x}) \to +\infty$. Таким образом для каждой точки $\mathbf{x}$ в пространстве признаков мы можем вычислить величину пропорциональную плотности числа точек. Применяя (9.14) на входном наборе данных $\mathbf{x}_i$ можно найти записи в наборе данных, которые расположены в областях с низкой плотностью точек, и сделать соответствующий вывод: либо исключить такие точки из набора данных, либо, наоборот, подробнее изучить.

9.2.4 Градиентное усиление над решающими деревьями

В предыдущих случаях все деревья строились независимо друг от друга, позволяя, кстати, почти неограниченно распараллеливать процесс построения леса между несколькими потоками исполнения. Такой подход часто называют *методом коммитетов*. Существует альтернативный способ составления ансамблей деревьев, известный как *усиление*[12] (см. Friedman 2001). Усиление предполагает, что искомая функция представляется рядом

$$f(\mathbf{x}) = \sum_{m=0}^{M} f_m(\mathbf{x}), \tag{9.15}$$

где слагаемые $f_m(\mathbf{x})$ строятся последовательно на основе $f_0(\mathbf{x}), ..., f_{m-1}(\mathbf{x})$.

Градиентное усиление использует следующий подход для решения задачи регрессии. В качестве $f_0(\mathbf{x})$ строится ограниченное по глубине решающее дерево, затем подсчитывается величина

$$\mathbf{r}^{(0)} \equiv \mathbf{f}^{(0)} - \mathbf{y} = \frac{1}{2}\nabla_{\mathbf{f}^{(0)}} \|\mathbf{f}^{(0)} - \mathbf{y}\|^2, \tag{9.16}$$

где вектор $\mathbf{y}$ обозначает все метки, а вектор $\mathbf{f}^{(0)}$ обозначает предсказание модели для всех точек учебного набора данных:

$$\mathbf{f}^{(0)} \equiv \begin{bmatrix} f_0(\mathbf{x}_1) \\ f_0(\mathbf{x}_2) \\ \vdots \\ f_0(\mathbf{x}_N) \end{bmatrix}. \tag{9.17}$$

Таким образом, вектор $\mathbf{r}^{(0)}$ по сути является вектором ошибок нашей модели на учебном наборе данных. Второе равенство в уравнении (9.16) объясняет название метода.

В качестве $f_1(\mathbf{x})$ строится следующее решающее дерево, но не для задачи регрессии y_i по признакам $\mathbf{x}_i$, а для регрессии $r_i^{(0)}$ по признакам $\mathbf{x}_i$. Иначе говоря, каждое следующее дерево учится исправлять ошибки предыдущего. А итоговое предсказание модели вычисляется по формуле (9.15), которой можно придать смысл последовательного уточнения предсказания с помощью коррекций всё более и более высокого порядка.

[12] *boosting*

9.3 Гиперпараметры и метрики качества

В предыдущих разделах было приведено несколько примеров алгоритмов машинного обучения для решения задач регрессии и классификации на основе ансамблей деревьев. Возникает важный и очевидный вопрос о том, как можно было бы характеризовать эффективность того или иного алгоритма. Во-первых, как это иногда бывает с методами машинного обучения, сами алгоритмы построены таким образом, что строго не очевидно, почему тот или иной метод вовсе приводит к успеху. В отличии, например, от метода наименьших квадратов (глава 3) или метода максимального правдоподобия (глава 6), где вначале постулируется базовый принцип, а затем все последующие действия выводятся из него формально и с достаточной математической строгостью, в случае с машинным обучением алгоритмы часто строятся по принципу проб и ошибок, а в качестве обоснования приводятся интуитивные рассуждения.

Во-вторых, многообразие доступных методов, очевидно, предполагает необходимость выбора некоторого наилучшего способа для решения задачи. Значит, необходимо иметь способ однородно охарактеризовать различные методы, примененные к решению одной и той же задачи.

В-третьих, чаще всего методы машинного обучения имеют некоторые настроечные параметры, которые не определяются непосредственно из данных, такие параметры принято называть *гиперпараметрами*. Например, в зависимости от выбранного метода гиперпараметрами могут быть предельная глубина решающего дерева, количество деревьев в ансамбле, доля данных, выбранных для бутстрапа, количество признаков, участвующих в поиске оптимального разбиения, архитектура нейронной сети (см. раздел 9.5) и т.п. Это означает, что даже в рамках одного метода необходимо сравнение случаев с разным выбором гиперпараметров.

Прежде всего, понятно, что алгоритмы машинного обучения могут быть охарактеризованы с точки зрения скорости работы: как в терминах асимптотической вычислительной сложности (см. раздел 2.2), так и с точки зрения измерения реального времени работы в конкретных условиях. Зачастую, время работы имеет немаловажное практическое значение, но верно и обратное — алгоритм, который быстро, но неправильно решает т задачу, как правило представляет мало интереса.

Чтобы охарактеризовать насколько хорошо тот или иной метод решает исходную задачу используются *метрики качества*. Метрики качества рассматривают модель, построенную с помощью методов машинного обучения, как инструмент, способный давать предсказания, при этом не вникая во внутреннее устройство модели, что и позволяет сравнивать между собой различные подходы. Например, для задачи регрессии в качестве метрики качества можно было бы выбрать среднеквадратичное отклонение[13] между

[13] *mean squared error (MSE)*

предсказанием модели и настоящими измерениями:

$$\mathrm{MSE} \equiv \frac{1}{N}\sum_{i=1}^{N}\|f(\mathbf{x}_i) - y_i\|^2, \tag{9.18}$$

где $\mathbf{x}_i$ — известные признаки, а y_i — заранее известный правильный ответ.

Такой выбор метрики качества для задачи регрессии не единственно возможный, в качестве альтернативы можно упомянуть, например, среднее абсолютное отклонение[14]:

$$\mathrm{MAE} \equiv \frac{1}{N}\sum_{i=1}^{N}|f(\mathbf{x}_i) - y_i|\,. \tag{9.19}$$

Конкретная метрика качества выбирается владельцем задачи исходя из представлений о том, какая именно проблема из реального мира решается в данном случае с помощью методов машинного обучения.

Метрики качества для задач классификации связаны с понятием *матрицы ошибок*[15], для случая бинарной классификации она выглядит следующим образом:

$$C \equiv \begin{bmatrix} \mathrm{TN} & \mathrm{FP} \\ \mathrm{FN} & \mathrm{TP} \end{bmatrix}, \tag{9.20}$$

и в каждой ячейке записано число событий, при использовании модели с набором данных с известными ответами:

- TN — число *истинно отрицательных*[16] событий, т.е. количество раз, когда модель назвала отрицательное событие отрицательным;
- FP — число *ложно положительных*[17] событий, т.е. количество раз, когда модель назвала отрицательное событие положительным;
- FN — число *ложно отрицательных*[18] событий, т.е. количество раз, когда модель назвала положительное событие отрицательным;
- TP — число *истинно положительных*[19] событий, т.е. количество раз, когда модель назвала положительное событие положительным.

Понятно, что сумма TN, FP, FN и TP равна полному числу данных в исследуемом наборе, потому-что кроме рассмотренных четырех вариантов других случаев быть не может. Видно, что колонки матрицы ошибок C соответствуют предсказанному классу, а строки соответствуют реальному классу. В случае, когда решается задача многоклассовой классификации, соответствующая матрица ошибок будет иметь столько же строк и колонок,

[14] *mean absolute error (MAE)*
[15] *confustion matrix*
[16] *true negative*
[17] *false positive*
[18] *false negative*
[19] *true positive*

сколько рассматриваемых классов. Матрица ошибок C не симметричная, потому-что в общем случае число ложно положительных и ложно отрицательных событий не обязаны совпадать, аналогично понятиям ошибок первого и второго рода из главы 8.

Подсчитав матрицу ошибок, можно ввести несколько метрик качества моделей для задач классификации, например, таких как *точность* (*accuracy*)[20] — доля правильно определенных классов:

$$\text{Accuracy} \equiv \frac{\text{TN} + \text{TP}}{N}. \tag{9.21}$$

Точность (*precision*) — доля правильно определённых положительных событий среди всех событий, названных положительными:

$$\text{Precision} \equiv \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{9.22}$$

при этом некоторое неопределенное число положительных событий может быть отклонено классификатором. *Полнота*[21] — доля найденных положительных событий по отношению к полному числу положительных событий:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{9.23}$$

при этом некоторое число отрицательных событий может быть принято классификатором за положительные. Понятно, что используя различные комбинации значений из матрицы ошибок можно построить и другие метрики качества.

Выбор конкретной метрики качества, как говорилось ранее, осуществляется владельцем задачи в зависимости от решаемой задачи. Например, в некотором эксперименте могут измеряться величины $\mathbf{x}_i$ от смеси двух распределений (как в разделе 6) и мы используем некоторый классификатор для отделения событий, принадлежащих каждому из распределений. Допустим, один класс событий — шумовые события, а другой класс — интересующие нас измерения характеристик частиц космического излучения. В одном случае, нас могут интересовать статистические характеристики измеряемых величин, например спектр энергии, тогда нам хотелось бы, чтобы выборка положительных событий была бы как можно меньше загрязнена неправильно классифицированными отрицательными (шумовыми) событиями, т.е. иметь высокую точность (precision), даже ценой того, что часть положительных событий оказалась отсеяна. В другом случае, например, когда исследуемые события очень редки (например, нас интересуют мюоны, порожденные солнечными нейтрино в толще Земли), может оказаться

[20] К сожалению, переводы многих терминов машинного обучения на русский язык в настоящее время всё ещё не устоялись. Так, в частности, под словом точность понимают две различных величины: *accuracy* и *precision*. Для устранения путаницы существуют сомнительные с точки зрения русского языка попытки перевода *accuracy* как *аккуратность*.

[21] *recall*

так, что нам не хотелось бы пропускать положительные события, т.е. иметь высокую полноту, пусть и ценой того, что в результирующей выборке окажется множество ложно положительных событий, которые, например, потребуют потратить время на дополнительную ручную проверку и анализ.

Вообще говоря, в большинстве случаев классификатор основан на сравнении некоторой непрерывной величины t с заданным порогом T. Аналогично тому, как устроены статистические критерии из главы 8. В роли такой величины выступает, например, доля ближайших точек, принадлежащих положительному классу в методе k ближайших соседей (см. введение главы 9), доля деревьев, относящих точку к положительному классу в методе случайного леса (см. раздел 9.2.2), или оценка условной плотности вероятности принадлежности к положительному классу в методе логистической регрессии (см. раздел 9.4). Понятно, что настраивая такой порог T для одного и того же классификатора можно получить различные показатели, например, точности и полноты. Достаточно очевидно, что при улучшении точности с помощью изменения порога T полнота будет ухудшаться, но «курс обмена» будет различный для различных классификаторов. Чтобы показать это формально, обозначим условные плотности вероятность величины t для события отрицательного класса и положительного класса как $p(t|z=0)$ и $p(t|z=1)$ соответственно, а условные плотности вероятности для вектора признаков $\mathbf{x}$ обозначим как $p(\mathbf{x}|z=0)$ и $p(\mathbf{x}|z=1)$. На практике ни одна из этих функций заранее точно не известна, а классификатор строит некоторое преобразование $t = f(\mathbf{x})$, формально связывая соответствующие $p(t|z)$ и $p(\mathbf{x}|z)$. Понятно, что если для какой-то области пространства признаков соответствующие $p(\mathbf{x}|z=0) \approx p(\mathbf{x}|z=1)$, то требовать от классификатора идеальной эффективности невозможно, потому-что исходные данные не предусматривают точного разделения.

Теперь мы можем записать теоретическое выражение для ожидаемой полноты, т.е. *доли истинно положительных событий*[22], в зависимости от заданного порога T:

$$\mathrm{TPR} = \int_T^\infty p(t|z=1)dt, \tag{9.24}$$

и *долю ложно положительных событий*[23]:

$$\mathrm{FPR} = \int_T^\infty p(t|z=0)dt. \tag{9.25}$$

Уравнения (9.24) и (9.25) задают параметрическую кривую на графике зависимости $\mathrm{TPR}(T)$ от $\mathrm{FPR}(T)$. Исторически сложилось, что эта кривая называется *рабочей характеристикой приёмника*[24]. Кривая во-первых, показывает соотношение между полнотой и долей ложно положительных событий. Во-вторых, позволяет ввести ещё одну популярную метрику качества классификатора, известную как *площадь под кривой ROC*[25]. Эта метрика

[22] *true positive rate (TPR)*
[23] *false positive rate (FPR)*
[24] *receiver operating characteristic (ROC)*
[25] *area under the curve (AUC)*

качества, очевидно, не зависит от настроечного параметра порога T и показывает насколько классификатор хорошо разделяет данные в принципе.

Совершенно очевидно, что конкретная величина метрики качества зависит не только от исследуемой модели, но и от используемой для вычисления этой метрики выборки данных. Напомним, что, независимо от решаемой задачи, наши признаки имеют некоторое распределение $p(\mathbf{x})$, а соответствующие метки — условное распределение $p(y|\mathbf{x})$. В рамках построения моделей нам доступна некоторая выборка из совместного распределения $\mathbf{x}_i$, y_i, которая обычно называется *учебной выборкой*[26] и используется для построения модели, т.е., например, для определения оптимальных разбиений при построении случайного леса.

Однако, эффективность модели мы хотим охарактеризовать для всех возможных данных, а не для конкретной подвыборки, иначе такая эмпирическая модель просто не может использоваться для предсказаний. Модель и нужна для того, чтобы достоверно предсказывать результат при новых условиях $\mathbf{x}$, для которых соответствующая метка y неизвестна. Иначе говоря, в идеальном мире мы должны были бы рассчитывать метрики типа (9.18), (9.19), или (9.20) не по выборке, а по настоящему распределению, т.е., например:

$$\mathrm{MSE} = \int d\mathbf{x}dyp(y|\mathbf{x})p(\mathbf{x})\|f(\mathbf{x}) - y\|^2, \tag{9.26}$$

однако использование такой формулы на практике невозможно, так как конкретный вид $p(\mathbf{x})$ и $p(y|\mathbf{x})$ не известен. Наивная замена интегрирования по распределению на суммирование по учебной выборке не корректна из-за эффекта *переобучения* при котором модель тем или иным способом просто запоминает учебную выборку (т.е. преобразует в свои коэффициенты, а в качестве предсказания выполняет поиск), но совершенно не обобщает закономерности, присутствующие в данных, и поэтому обладает слабой предсказательной силой. Поэтому для оценки метрик качества модели должна использоваться отдельная выборка, называемая *тестовой выборкой*, но полученная из того же распределения, что и учебная. На практике, все доступные данные случайным образом разделяются на две подвыборки, и одна часть используется для обучения, а вторая откладывается до момента оценки качества модели.

Кроме того, осторожность следует соблюдать в процессе подбора гиперпараметров модели. Процедура подбора гиперпараметров состоит в последовательном переборе значений гиперпараметров и оценке метрики качества модели с использованием так называемой *валидационной выборки*, которая имеет такие же статистические свойства как учебная и тестовая выборки. Необходимость в отдельной выборке обусловлена тем, что процесс перебора значений гиперпараметров, независимо от того выполняется ли он вручную или с помощью специализированного алгоритма, представляет собой вариант алгоритма оптимизации, следовательно, может сложиться такая ситуация, в которой гиперпараметры подобраны таким образом, что

[26] *training set*

модель *случайно* имеет высокую эффективность для валидационной выборки. Поэтому необходимо использовать тестовую выборку только для окончательной характеризации качества полученной модели с подобранными значениями гиперпараметров. На практике, все доступные данные случайным образом разделяются на три подвыборки.

9.4 Логистическая регрессия

Помимо моделей основанных на деревьях, понятние машинное обучение включает в себя множество разнообразных методов, наибольшую известность среди которых в последнее время, пожалуй, имеют методы нейронных сетей (см. раздел 9.5). Перед тем как переходить к их рассмотрению имеет смысл изучить другой метод, называемый методом *логистической регрессии*. Как будет видно далее, логистическая регрессия представляет собой один из простейших вариантов нейронной сети, который, тем не менее, имеет очевидный способ обобщения.

Метод логистической регрессии используется для решения задачи классификации. Далее сначала рассмотрим случай бинарной классификации, а затем легко обобщим его на случай нескольких классов.

Вспомним, что в разделе 6.1 рассматривались модели со скрытыми переменными и смеси распределений. Перепишем выражение (6.27) для условной вероятности принадлежности измерений тому или иному классу при условии известного вектора признаков $\mathbf{x}$:

$$p(\{z=j\}\,|\mathbf{x},\theta') = \frac{p(\mathbf{x}|\,\{z=j\}\,,\theta')p\,\{z=j|\theta'\}}{\sum_{k\in\{0,1\}} p(\mathbf{x}|\,\{z=k\}\,,\theta')p\,\{z=k|\theta'\}}. \tag{9.27}$$

В отличии от раздела 6.1, теперь мы считаем, что в рамках учебной выборки переменная y, обозначающая принадлежность измерений тому или иному классу, известна, т.е. больше не является скрытой. Параметры θ' и аналитический вид $p(\mathbf{x}|z,\theta')$ нам неизвестны и не интересны — в рамках данного подхода нас интересует только возможность вычислять (9.27) для любого вектора $\mathbf{x}$, получая вероятность принадлежности события определённому классу.

Для этого преобразуем (9.27) для $j=0$ в следующий вид, разделив знаменатель на числитель:

$$p(\{z=0\}\,|\mathbf{x},\theta') = \frac{1}{1+\exp\left(-\ln\left(\frac{p(\mathbf{x}|\{z=0\},\theta')p\{z=0|\theta'\}}{p(\mathbf{x}|\{z=1\},\theta')p\{z=1|\theta'\}}\right)\right)}, \tag{9.28}$$

для удобства, обозначим логарифм в этом выражении новой переменной $v(\mathbf{x})$, и аппроксимируем эту неизвестную функцию линейным разложением:

$$p(\{z=0\}\,|\mathbf{x},\theta') = \frac{1}{1+\exp(-v(\mathbf{x}))}, \tag{9.29}$$

$$v(\mathbf{x}) = (\mathbf{x},\mathbf{w}) + b, \tag{9.30}$$

где $\mathbf{w}$ и b — некоторые новые неизвестные параметры, которые уже не зависят от конкретного распределения. Функцию вида (9.29) называют *логистическим сигмоидом*, она монотонно возрастает от 0 до 1 и определена для всех значений v. Корректность приближения (9.30) можно оправдать, рассматривая два нормальных распределения $p(\mathbf{x}|z,\theta')$ с одинаковыми матрицами ковариации, в таком случае линейность выражения может быть проверена строго, т.е. новые параметры $\mathbf{w}$ и b выразятся через традиционные параметры многомерного нормального распределения. В других случаях, (9.29) является эмпирической моделью.

Необходимо отыскать оценки новых параметров $\mathbf{w}$ и b, которые, в отличии от раздела 6.1, больше не представляют для нас самостоятельного интереса. Проще всего выполнить оценку с помощью метода максимального правдоподобия, так как для учебной выборки нам одновременно известны и метки y_i, определяющие принадлежность к классу, и вектора признаков x_i. Запишем отрицательный логарифм правдоподобия, который в этом случае называют *перекрёстной энтропией*:

$$\begin{aligned}-\ln p\left\{z_1 = y_1, z_2 = y_2, ..., z_N = y_N | \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N, \theta\right\} = \\ = -\sum_{i=1}^{N} \left(y_i \ln p(\left\{z = 1\right\}|\mathbf{x}_i, \theta) + (1 - y_i) \ln\left(1 - p(\left\{z = 1\right\}|\mathbf{x}_i, \theta)\right)\right), \quad (9.31)\end{aligned}$$

где θ обозначает параметры $\mathbf{w}$ и b, а $y_i \in \{0, 1\}$. Оценка параметров находится с помощью численных методов оптимизации (9.31) в соответствии с методом максимального правдоподобия (см. главу 6).

В случае многоклассовой классификации вместо выражения (9.27) получается следующее:

$$p\left\{z = j|\mathbf{x}, \theta'\right\} = \frac{\exp(v_k(\mathbf{x}))}{\sum_j \exp(v_j(\mathbf{x}))}, \quad (9.32)$$

$$v_j(\mathbf{x}) \equiv \ln p(\mathbf{x}|z = j, \theta) p\left\{z = j|\theta\right\}. \quad (9.33)$$

Эта функция названа *нормированной экспонентой* или *softmax*. Затем, каждая функция $v_j(\mathbf{x})$ приближается линейной зависимостью с неизвестными параметрами $\mathbf{w}_j$ и b_j, которые находятся исходя из метода максимального правдоподобия, минимизацией перекрёстной энтропии:

$$\begin{aligned}-\ln p\left\{z_1 = y_1, z_2 = y_2, ..., z_N = y_N | \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N, \theta\right\} = \\ = -\sum_{i=1}^{N} \sum_{j=1}^{K} h_{ij} \ln p(\left\{z = y_i\right\}|\mathbf{x}_i, \theta), \quad (9.34)\end{aligned}$$

где h_{ij} обозначает представление метки класса в виде *унитарного кода*[27]:

$$h_{ij} \equiv \begin{cases} 1 & \text{если } y_i = j, \\ 0 & \text{иначе.} \end{cases} \quad (9.35)$$

[27] *one-hot encoding*

9.5 Нейронные сети

9.5.1 Нейронные сети прямого распространения

Очевидный недостаток метода логистической регрессии (см. раздел 9.4) состоит в том, что линейное приближение не всегда хорошо работает для реальных распределений признаков. Для улучшения модели требуется предусмотреть возможность нелинейной апроксимации $v(\mathbf{x})$. Метод нейронный сетей строит такую нелинейную апроксимацию методом композиции функций (т.е. построения вложенных функций) следующим образом. Представим, что теперь $v(\mathbf{x})$ задаётся следующим образом:

$$v(\mathbf{x}) = (\mathbf{w}_2, \mathbf{z}(\mathbf{x})) + b_2, \tag{9.36}$$

где $\mathbf{z}(\mathbf{x})$ в свою очередь определяется как

$$\mathbf{z} = \varphi\left((\mathbf{w}_1, \mathbf{x}) + b_1\right). \tag{9.37}$$

Здесь φ обозначает поэлементное применение к аргументу-вектору нелинейного преобразования, называемого *функцией активации*. Для простоты можно считать, что φ обозначает хорошо известную нам функцию логистическая сигмоида:

$$\left(\varphi(\mathbf{v})\right)_i = \frac{1}{1 + \exp(-v_i)}, \tag{9.38}$$

однако на практике используются и другие варианты, такие, например, как гиперболический тангенс

$$\left(\varphi(\mathbf{v})\right)_i = \mathrm{th}(v_i), \tag{9.39}$$

или даже линейный выпрямитель[28]:

$$\left(\varphi(\mathbf{v})\right)_i = \max\left\{0, v_i\right\}. \tag{9.40}$$

Уравнения (9.29), (9.44) и (9.37) вместе определяют простую модель нейронной сети прямого распространения с одним скрытым слоем:

$$p(\{z = 0\}\,|\mathbf{x}, \theta') = \varphi((\mathbf{w}_2, \varphi\left((\mathbf{w}_1, \mathbf{x}) + b_1\right)) + b_2), \tag{9.41}$$

где θ' обозначает параметры $\mathbf{w}_1$, $\mathbf{w}_2$, b_1 и b_2. На практике подобная запись применяется редко и нейронные сети принято изображать графически. В терминологии нейронных сетей каждый компонент вектора признаков $\mathbf{x}$, вектора $\mathbf{z}$ и вероятность $p\left\{z = 0|\mathbf{x}, \theta'\right\}$ принято называть *нейроном*. Нейроны объединяются в *слои*, и тогда вектор $\mathbf{x}$ называют *входным слоем*, вектор $\mathbf{z}$ *скрытым слоем*, а предсказываемую вероятность — *выходным слоем*. В случае бинарной классификации выходной слой состоит из одного нейрона, в случае многоклассовой классификации выходной слой чаще всего представляется с помощью вектора вероятностей точно так же как в случае

[28] ReLU (Rectified linear unit)

логистической регрессией. Понятно, что число скрытых слоёв и их размер являются гиперпараметрами.

Параметры θ' называют *весами* и они всё ещё подлежат определению исходя из учебной выборки данных и минимизации выбранной *функции потерь*. Для задачи классификации функция потерь выбирается в виде (9.31) или (9.34) в зависимости от рассматриваемого числа классов. Процесс оценки неизвестных весов с помощью алгоритмов численной оптимизации называют *обучением нейронной сети*.

Несмотря на то, что мы оттолкнулись от метода логистической регрессии, нейронные сети могут применяться не только для решения задач классификации, но и, например, для задач регрессии. В последнем случае, во-первых, требуется выбрать функцию активации для выходного слоя таким образом, чтобы она соответствовала ожидаемому диапазону меток y, либо отнормировать данные соответствующим образом. Во-вторых, в качестве функции потерь требуется выбрать функцию адекватную задаче, например среднеквадратичное отклонение:

$$E(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i|\theta)\right)^2, \tag{9.42}$$

где y_i обозначают метки учебной выборки, соответствующие признакам $\mathbf{x}_i$, а $f(\mathbf{x}|\theta)$ обозначает всю нейронную сеть целиком, как некоторую функцию, параметры θ определяются в процессе обучения. В таком виде метод нейронных сетей соответствует задаче нелинейных наименьших квадратов (5.24).

Нейронные сети применяются и для решения задач машинного обучения без учителя, например, для задачи уменьшения размерности. Популярной архитектурой, предназначенной для этого, является *автокодировщик* или *самоассоциативная нейронная сеть*. Можно сказать, что автокодировщик является нелинейным обобщением анализа главных компонент (см. главу 4). Автокодировщик представляет собой сеть прямого распространения, в которой входной и выходной слои имеют одинаковый размер, а один из скрытых слоёв имеет меньший размер. Тогда первая половина сети называется кодировщиком, а вторая декодировщиком. Функция потерь выбирается в следующем виде (сравните с выражением (4.18)):

$$E(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{f}(\mathbf{x}_i|\theta)\|^2, \tag{9.43}$$

где метки y_i в рассмотрении не участвуют, так как решается задача обучения без учителя, т.е. без меток. Фактически, мы пытаемся апроксимировать тождественное преобразование, однако, малый размер скрытого слоя в середине сети мешает сделать это тривиальным способом. По принципу построения нейронных сетей выходной слой $\mathbf{f}(\mathbf{x}_i|\theta)$ неявно зависит от скрытого слоя меньшей размерности $\mathbf{z}$, который в свою очередь зависит от входных

данных. Таким образом кодировщик должен научится упаковывать представленный на вход вектор $\mathbf{x}$ в пространство меньшей размерности с потерей минимального количества информации, а декодировщик нужен чтобы кодирощик мог обучаться. В каком-то смысле декодировщик проверяет, что информация может быть преобразована в исходную форму.

После обучения первую часть нейронной сети, кодировщик, используют отдельно для решения задачи понижения размерности. Такая модель может использоваться как готовый блок для решения других задач, например классификации или регрессии. Вместо построения нейронной сети, которая принимает на вход исходный вектор признаков, строится нейронная сеть, которая принимает на вход сжатое представление. Такая сеть будет иметь меньше параметров, следовательно требовать меньших вычислительных ресурсов и данных для своего обучения. Автокодировщик вцелом способен решать задачу фильтрации случайных шумов, подобно анализу главных компонент.

9.5.2 Обучение на основе коррекции ошибок. Обратное распространение ошибки

Рассмотрим важный вопрос о способе выполнения численной оптимизации над функциями, заданными в виде нейронных сетей. Любой метод, кроме всего прочего, должен быть практически реализуем, т.е. все величины должны быть рассчитаны за разумное время, иначе от него не слишком много практической пользы. Понятно, что рассчёт предсказания нейронной сети требует примерно одинаковое количество умножений и сложений, пропорциональное числу параметров модели, что кажется достаточно разумным. В таком случае рассчёт предсказания выполняется послойно — от входного слоя к выходному.

Для эффективного выполнения численной оптимизации требуется рассчитать градиент функции потерь по искомым параметрам. К счастью, оказывается, что есть достаточно эффективный способ вычисления градиента, называемый методом обратного распространения ошибки. Пусть каждый слой задаётся следующими уравнениями:

$$\mathbf{v}^{(l)} = W_l \mathbf{z}^{(l)} + \mathbf{b}_l, \tag{9.44}$$

$$\mathbf{z}^{(l+1)} = \varphi\left(\mathbf{v}^{(l)}\right). \tag{9.45}$$

Для однородности изложения $\mathbf{z}^{(0)}$ обозначает входной слой $\mathbf{x}$, а соответствующий $\mathbf{z}^{(n+1)}$ обозначает выходной слой нейронной сети. Пусть задана некоторая функция потерь $E(\mathbf{z}^{(n+1)})$, например в виде (9.31), (9.34), (9.42) или (9.43), требуется вычислить частные производные этой функции по искомым параметрам: матрицам W_1, W_2, ... W_n, и векторам $\mathbf{b}_1$, $\mathbf{b}_2$, ... $\mathbf{b}_n$.

Прежде всего заметим, что функция потерь зависит от некоторого параметра только через значение соответствующего слоя, т.е., например, спра-

ведливо:

$$\frac{\partial E}{\partial W_{ij}^{(l)}} = \frac{\partial E}{\partial v_i^{(l)}} \frac{\partial v_i^{(l)}}{\partial W_{ij}^{(l)}} = \frac{\partial E}{\partial v_i^{(l)}} z_j^{(l)}, \tag{9.46}$$

$$\frac{\partial E}{\partial b_i^{(l)}} = \frac{\partial E}{\partial v_i^{(l)}} \frac{\partial v_i^{(l)}}{\partial b_i^{(l)}} = \frac{\partial E}{\partial v_i^{(l)}}. \tag{9.47}$$

Иначе говоря, рассчитав частные производные функции потерь по значениям слоёв, мы могли бы сравнительно легко пересчитать таки производные в нужные производные по параметрам. Для этого заметим, что функция потерь зависит от значений на слое l только косвенно через значение слоя $l+1$:

$$\frac{\partial E}{\partial v_i^{(l)}} = \sum_k \frac{\partial E}{\partial v_k^{(l+1)}} \frac{\partial v_k^{(l+1)}}{\partial v_i^{(l)}} = \sum_k \frac{\partial E}{\partial v_k^{(l+1)}} w_{ki}^{(l)} \phi' \left(v_i^{(l)} \right). \tag{9.48}$$

Если представить, что при подсчёте функции потерь соответствующие значения слоёв будут временно записаны в память, то полученные уравнения позволяют рассчитать значения всех производных рекуррентно от выходного слоя к входному. Этим и объясняется название метода обратного распространения ошибки: рассчёт предсказания производится пошагово в прямом направлении, а производной — в обратном. К счастью, популярные пакеты для работы нейросетевыми моделями выполняют автоматическое дифференцирование, и избавляют от необходимости программирования вычисления градиента, так как градиент вычисляется автоматически для любой заданной архитектуры сети.

При обучении нейронных сетей популярно применение алгоритмов типа стохастического градиентного спуска (см. раздел 5.4.1), при котором на каждом шаге используется лишь малая часть доступных данных. Такой способ обучения называется последовательным. Для каждой отдельной записи или небольшого набора, называемого *пакетом*[29], вычисляется оценка градиента, а затем веса подравляются на некоторую величину в направлении антиградиента:

$$w^{(k+1)} = w^{(k)} - \nu \nabla E, \tag{9.49}$$

где ν называется *скоростью обучения*. Когда весь набор данных последовательно использован для оценки градиентов, говорят, что прошла одна эпоха обучения. Одна эпоха соответсвовала бы одному шагу оптимизации при использовании пакетного обучения, т.е. когда градиент вычислялся бы на всех доступных данных сразу. Зависимость значения функции потерь от эпохи обучения называется *кривой обучения*.

Отдельно стоит упомянуть проблему выбора начальных значений весов. На практике они выбираются случайным образом. Это связано с тем, что пространство весов обладает большим числом симметрий, так например, все нейроны в рамках одного полносвязного слоя равноправны, поэтому

[29] *batch*

задание весам одинаковых значений (например нулевых) приведёт к тому, что в процессе оптимизации эти значения будут обновляться на одинаковую величину.